

Тернопільський національний технічний університет імені Івана Пулюя

**Кафедра автоматизації
технологічних
процесів та виробництв**

Методичні вказівки до виконання

Лабораторних робіт
з курсу "**Основи системного програмування**"

Тернопіль, 2017

Методичні вказівки до виконання лабораторних робіт з курсу “Основи системного програмування” / Уклад. Коноваленко І.В.– Тернопіль: ТНТУ, 2017.

Укладач: к.т.н., доц. каф. автоматизації технологічних процесів і виробництв Коноваленко І.В.

Рецензент к.т.н., доц. каф. автоматизації технологічних процесів і виробництв Савків В.Б.

Методичні вказівки розглянуті та затверджені на засіданні кафедри автоматизації технологічних процесів і виробництв.

Протокол №3 від 19 вересня 2017 р.

Вступ

1. Системне програмування

Системне програмування представляє собою вид програмування, що використовує пряму взаємодію прикладної програми з операційною системою (ОС), враховуючи при цьому особливості її архітектури та принципи роботи. Знання принципів взаємодії прикладної програми з операційною системою необхідне, наприклад, при розробці програмного забезпечення для роботи з апаратним забезпеченням (плати ЦАП/АЦП, різноманітні інтерфейси тощо). При цьому слід зауважити, що різні операційні системи з таким обладнанням можуть працювати по-різному, в залежності від особливостей їх роботи.

Зараз існує багато різноманітних типів операційних систем, які відрізняються областю використання, апаратними платформами та методами реалізації. Це, в свою чергу, обумовлює і значні функціональні відмінності цих ОС. Операційна система містить комплекс взаємозв'язаного програмного забезпечення, яке діє як інтерфейс між прикладними програмами та користувачами з однієї сторони, і апаратною ЕОМ з іншої сторони. У відповідності до цього ОС виконує такі дві групи функцій:

- надання користувачеві або програмістові замість реальної апаратури розширеної віртуальної машини, з якою зручніше працювати і яку легше програмувати;
- підвищення ефективності використання ЕОМ шляхом раціонального управління його ресурсами у відповідності до певного критерію.

Функції операційної системи, як правило, групують у відповідності до типу локальних ресурсів, якими керує ОС, або у відповідності із специфічними завданнями, які виконує ОС. Такі групи функцій називають підсистемами. Найважливішими підсистемами управління ресурсами є підсистеми керування процесами, пам'яттю, файлами та зовнішніми пристроями. Загальними для всіх ресурсів є підсистеми інтерфейсу користувача, захисту даних та адміністрування.

Знання основних принципів системного програмування є надзвичайно корисним і при розробці звичайних прикладних програм, які не здійснюють прямої взаємодії з апаратним забезпеченням, так як знання внутрішньої архітектури програми дозволяє не лише краще розуміти спосіб її функціонування, але й використовувати при її роботі різноманітні нестандартні засоби, які при застосуванні середовищ візуального програмування (таких як **Borland Delphi**, **Borland C++ Builder**) є найчастіше скриті.

Оскільки різні операційні системи відрізняються як внутрішньою архітектурою, так і способами взаємодії з апаратним та програмним забезпеченням, то принципи системного програмування для різних ОС є відмінними. Даний лабораторний курс призначений для вивчення особливостей програмування в операційних системах **Microsoft Windows**. Слід пам'ятати, що розробка прикладних програм, які здійснюватимуть одні і ті ж дії на різних ОС, може суттєво відрізнятися.

2. Лабораторний курс з основ системного програмування

Метою курсу “Основи системного програмування” є формування у студентів знань про принципи функціонування операційних систем **Microsoft Windows**, взаємодію прикладних програм з ОС та принципи розробки прикладних програм для операційної системи **Windows**.

В результаті вивчення дисципліни студент повинен знати основні принципи роботи ОС **Windows**, основні аспекти побудови прикладних програм для різних версій ОС та для різних платформ.

В результаті вивчення дисципліни студент повинен вміти розробляти прикладні програми, які використовують особливості ОС **Windows**.

Для отримання якісного рівня підготовки фахівців необхідно мати базовий рівень з таких попередніх і паралельно-попередніх дисциплін: основи програмування, об'єктно-орієнтоване програмування, мікропроцесорна техніка, вища математика.

Завданням даного курсу є:

- вивчення архітектури ОС **Windows**;
- вивчення основних функцій **Windows API**;
- вивчення структури прикладної програми;
- вивчення методів взаємодії прикладної програми з ОС та з іншою програмою;
- вивчення принципів розробки прикладних програм;
- вивчення принципів взаємодії прикладної програми з апаратним забезпеченням ЕОМ;
- ознайомлення з основними командами мови assembler для здійснення низькорівневого доступу до ресурсів ЕОМ.

Вимоги до оформлення лабораторних робіт

Для зарахування лабораторної роботи студент повинен представити:

1. Проект, створений у **Borland Delphi**, **Borland C++ Builder** або **Microsoft Visual C++** у якому виконується завдання згідно заданого викладачем варіанту на лабораторну роботу. В проектах не повинні використовуватися компоненти **VCL**, якщо це спеціально не обумовлено у завданні.
2. Звіт про виконання лабораторної роботи. Звіт повинен бути виконаний на стандартних аркушах формату А4.

Звіт про виконання лабораторної роботи повинен містити:

1. Тему роботи та завдання згідно варіанту.
2. Короткий опис створеної згідно завдання програми, включаючи:
 - Блок-схему алгоритму, використаного для вирішення завдання, та її короткий опис;
 - Опис використаних ідентифікаторів та змінних;
 - Сценарії спроектованих діалогових вікон і призначення елементів керування у них;
 - Текст проекту і самостійно спроектованих модулів.

Лабораторна робота №1

Тема: Створення простого вікна засобами Windows API.

Мета: Вивчення основних принципів створення прикладних програм за допомогою Windows API. Ознайомлення з принципом взаємодії вікна з операційною системою через систему повідомлень.

1. Принципи функціонування прикладних програм в середовищі ОС Windows

1.1. Система повідомлень в ОС Windows

Багато рис операційної системи **Windows** зобов'язані своєю специфікою тому факту, що вона є системою, основою на повідомленнях. Такий підхід до програмування був вперше реалізований в операційній системі для комп'ютерів **Macintosh** і полягає він у тому, що поведінка (і реакція) програми визначається зовнішніми подіями. На відміну від традиційного "послідовного" програмування, програмування для **Windows** є об'єктно-орієнтованим. І, не зважаючи на те, що для цього можуть бути використані зовсім не об'єктно-орієнтовані мови, саму систему **Windows** можна представити у виді набору об'єктів, з якими взаємодіє програма. Найважливішим таким об'єктом є вікно. Воно першим з'являється при запуску будь-якої прикладної програми (чи самої системи) й останнім зникає при завершенні роботи. У процесі роботи **Windows** повідомляє прикладній програмі (точніше, вікну прикладної програми) про деяку подію за допомогою повідомлення, яке описує зміну, що відбулася в навколишньому середовищі (натискання клавіші клавіатури, кнопки миші, рух вказівника та інші). Крім цього, механізм повідомлень для прикладних програм – єдиний спосіб "спілкування" з їхніми вікнами. Відбувається це не прямо, а через операційну систему, яка, отримавши повідомлення, передає його потрібному об'єкту (вікну). Повідомлення використовуються для настройки, зміни і діагностики стану всіх вікон. Воно є подією, на яку, при необхідності, повинна реагувати прикладна програма. Будь-яке повідомлення пов'язується з конкретним вікном, кожне з яких, у свою чергу, має власну віконну функцію, тобто спеціальну функцію, яка відповідає за обробку вхідних повідомлень. При цьому всі повідомлення, джерелом яких служать апаратні засоби, система переводить у визначену структуру даних – первинну вхідну чергу.

У ранніх версіях **Windows (Windows 3.x)** система підтримувала лише одну чергу повідомлень, з якої їх і вибирали прикладні програми. У цих системах реалізована так звана невитісняюча багатозадачність. Проте при цьому виникали проблеми при збої в активній прикладній програмі, коли його невибрані з черги повідомлення не давали можливості іншим програмам отримати "свої" повідомлення.

У 32-розрядних операційних системах **Windows** реалізовано метод підтримки індивідуальних черг повідомлень, який називається десинхронізацією вводу. При цьому нові повідомлення лише ненадовго попадають у первинну вхідну чергу 32-розрядної прикладної програми, після чого перенаправляються до черги конкретного потоку. Спрощено механізм можна описати таким чином. На початку своєї роботи 32-розрядна програма отримує від системи єдину чергу повідомлень, що пов'язується з його первинним потоком. І навіть якщо процес заводить ще один потік, система не відразу створює для нього ще одну чергу. Така черга створюється лише тоді, коли додатковий потік даної програми вперше направить у систему запит до черги. Якщо потоку черга повідомлень не потрібна, то система взагалі її не створює, щоб не витратити ресурси.

Кожне вікно в **Windows** має спеціальну функцію, яка викликається для обслуговування повідомлень. Операційна система лише посилає повідомлення вікну, а його віконна функція повинна забезпечити бажану реакцію на це повідомлення. Протягом всього часу існування вікно діє як автономний об'єкт, який "живе" своїм власним життям.

Для посилення повідомлень об'єктам системи, якщо відомі їхні ідентифікатори, використовується функція **SendMessage**. Вона відсилає повідомлення певному об'єкту (можливо, з додатковими даними), чекає на реакцію цього об'єкта на повідомлення (як правило у виді числа), і

повертає результат системі. Об'єктом може бути не лише вікно прикладної програми, документа, діалогове вікно і т.д., але і будь-який елемент керування, наприклад, кнопка. Повідомлення, передане за допомогою функції **SendMessage**, практично негайно обробляється віконною процедурою, остання отримує його не з черги, а прямо. Крім цього, **Windows** дає можливість прикладним програмам можливість послати повідомлення об'єкту, використовуючи чергу. Здійснюється це за допомогою функції **PostMessage**. У цьому випадку прикладна програма продовжує виконуватися, не чекаючи реакції на повідомлення.

1.2. Інтерфейс прикладного програмування

Інтерфейс прикладного програмування (**Application Programming Interface, API**) – це набір необхідних функцій, за допомогою яких будь-яка прикладна програма може взаємодіяти з операційною системою.

В межах операційної системи **Windows** з'явилося три таких **API** (фактично їх більше). Найперший з них **Win16** – інтерфейс 16-розрядних операційних систем **Microsoft**. Спроба збільшити розрядність існуючої версії привела до появи **Win32s** – 32-розрядного інтерфейсу для 16-розрядних систем. З появою 32-розрядних систем **Windows** з'явився і 32-розрядний інтерфейс – **Win32**. Більшість функцій **API** повертають код виходу, а значення параметрів передаються через вказівники, задані параметрами.

API – це сполучна ланка між прикладними програмами й операційною системою. **Win32 API** містить біля двох тисяч функцій і кілька сотень повідомлень, макросів і визначених констант. Це приводить до того, що з одного боку, програміст не має необхідності дублювати частини операційної системи у своїй програмі, як це обов'язково відбувається при створенні прикладних програм **MS-DOS**, з іншого боку, такі широкі можливості серйозно ускладнюють процес навчання.

Win32 API складається з трьох головних компонентів: **Kernel**, **User** і **GDI**, що забезпечують інтерфейс із базовою операційною системою (**Kernel**), керування вікнами і прикладними програмами (**User**), а також підтримку графіки (**GDI**).

1.3. Структура прикладної програми з точки зору ОС

Будь-яка прикладна програма **Windows**, яка має інтерфейс із користувачем (за винятком консольних прикладних програм), складається з двох основних частин: функції **WinMain**¹ (стандартна назва для мови **C** та **C++**) і функції вікна. Функція **WinMain** прикладної програми, містить три блоки: блок реєстрації класу вікна, блок створення головного вікна програми, та циклу обробки повідомлень.

Стандартний цикл обробки повідомлень працює протягом всього часу виконання програми. Кожна його ітерація приводить до витягування одного повідомлення з черги повідомлень активного потоку. За це відповідає функція **GetMessage**, другий аргумент якої (**0**) говорить про те, що слід обробляти всі повідомлення.

Після того як повідомлення вибране з черги повідомлень, воно передається у функцію **TranslateMessage**. Її основне призначення полягає у виклику драйвера клавіатури **Windows** для перетворення віртуальних кодів клавіш у **ASCII**-значення, що ставляться в чергу програмних подій у виді повідомлення **WM_CHAR**. Це дозволяє програмі відрізнити, наприклад, "A" від "a" без аналізу стану клавіші регістра.

Остання функція циклу – **DispatchMessage** – передає дані про повідомлення для обробки у відповідну віконну процедуру. Після того як повідомлення передане, знову викликається функція **GetMessage**, щоб взяти з черги наступне повідомлення (при його наявності). За винятком **WM_QUIT**, кожне повідомлення змушує **GetMessage** повернути значення **TRUE**. Приймаючи повідомлення **WM_QUIT**, програма виходить з циклу обробки і завершує свою роботу. Такий (чи подібний) цикл безперервно повторюється протягом усього строку існування прикладної програми. Він складає основу і суть будь-якої програми **Windows**.

¹ У стандарті мови **Object Pascal** цю функцію можна назвати по-своєму. При використанні **VCL Delphi** вона створюється автоматично, без втручання програміста.

2. Приклад простої програми, написаної з використанням засобів Windows API

У пункті 4 даних методичних вказівок приведено текст програми, яка створює головне вікно з двома кнопками, одна з кнопок призначена для виходу з програми, інша – для відображення інформації про неї. Особливістю даної програми є те, що вона не містить ні однієї стандартної форми **Delphi** – всі вікна створюються динамічно, в процесі роботи. Ще одна її "особливість" – розмір скомпільованого виконавчого файлу становить всього 10 Кб, в той час як така ж проста програма з одним вікном, створена з використанням бібліотеки візуальних компонентів (**VCL**) **Delphi**, має розмір понад 300 Кб ²! Крім цього, програми, що не використовують **VCL**, працюють швидше. Ці риси (великі виконавчі модулі та зменшена швидкодія) характерні для всіх систем розробки програм, які використовують потужні об'єктно-орієнтовані бібліотеки (**VCL** у **Borland Delphi**, **MFC** у **Microsoft Visual C++** та інші). Проте суттєвою перевагою об'єктно-орієнтованих систем є те, що використання таких бібліотек дозволяє в десятки раз скоротити тривалість розробки складних прикладних програм.

Оскільки у даній програмі не використовується **VCL**, її можна набрати у будь-якому текстовому редакторі (наприклад, "Блокнот" **Windows**), і зберегти як звичайний текст з розширенням **.DPR**. Якщо ж її створювати в середовищі **Delphi**, можна поступити так:

- 1) Створити новий проект (**File\New\Application**).
- 2) Оскільки **Delphi** автоматично для нового проекту створює форму головного вікна, то треба видалити з проекту модуль цієї форми (і її саму) за допомогою команди меню **Project\Remove from Project...**
- 3) Перейти до файлу проекту (**ProjectView Source**) та ввести текст програми.

Головний блок коду програми має наступний вид:

```
const AppName='API Window';
var AMessage : TMsg;
    hWindow   : HWND;
.....
begin
  if not WinRegister then
  begin
    MessageBox(0,'Клас вікна не зареєстровано',AppName,MB_OK);
    Exit;
  end;
  hWindow:=WinCreate;
  if hWindow=0 then
  begin
    MessageBox(0,'Не вийшло створити вікно.',AppName,MB_OK);
    Exit;
  end;
  while GetMessage(AMessage,0,0,0) do
  begin
    TranslateMessage(AMessage);
    DispatchMessage(AMessage);
  end;
  Halt(AMessage.wParam);
end.
```

Спочатку за допомогою функції **WinRegister** реєструється клас головного вікна програми. Після цього за допомогою функції **WinCreate** створюється саме вікно, і його ідентифікатор (**handle**) запам'ятовується у глобальній змінній **hWindow**. У випадку неуспішного виконання однієї з цих операцій здійснюється вихід з програми, так як ніяка робота програми без головного вікна неможлива. Далі програма містить цикл обробки повідомлень, який приймає повідомлення, що надходять до неї, перетворює їх та посилає до віконної функції головного вікна. Цикл продовжується доти, поки не буде закрито головне вікно, після чого програма за допомогою функції **Halt** передає операційній системі (**Windows**) значення **wParam** останнього повідомлення у

² При використанні **Borland Delphi 6 Enterprise**.

виді коду виходу. **Halt** є загальноприйнятим викликом для закінчення роботи програми. Такий спосіб завершення програми гарантує, що її виконання припиниться із заданим кодом виходу. Проте, якщо опустити звертання до функції **Halt**, програма все одно коректно завершить свою роботу.

Для реєстрації класу вікна в середовищі операційної системи використовується функція **WinRegister**³:

```
function WinRegister: boolean;
var WindowClass: TWndClass;
begin
  with WindowClass do begin
    Style:=CS_HREDRAW or CS_VREDRAW;
    lpfnWndProc:=@WindowProc;
    cbClsExtra:=0;
    cbWndExtra:=0;
    hInstance:=HInstance;
    hIcon:=LoadIcon(0, IDI_APPLICATION);
    hCursor:=LoadCursor(0, IDC_ARROW);
    hbrBackGround:=COLOR_WINDOW;
    lpszMenuName:=nil;
    lpszClassName:=AppName;
  end;
  Result:=RegisterClass(WindowClass)<>0;
end;
```

Структура⁴ **TWndClass** містить значну частину інформації про тип вікна, яке реєструється. У таблиці №1.1 приведено елементи цієї структури та описано їх призначення.

Таблиця №1.1

Елемент	Призначення
Style	Задає стиль класу. Стилі можуть поєднуватись за допомогою оператора or . Стиль CS_HREDRAW означає, що вікно буде автоматично перемальовуватися, якщо вікно переміщається або змінюється його ширина. Стиль CS_VREDRAW дає той же ефект при зміні висоти вікна.
lpfnWndProc	Вказівник на віконну функцію (або, її ще називають процедурою вікна).
cbClsExtra	Задає кількість додаткових байтів, які система виділяє для структури класу вікна.
cbWndExtra	Задає кількість додаткових байтів, які система виділяє для екземпляру вікна. Якщо програма використовує дану структуру для реєстрації діалогового вікна, створеного з використанням директиви CLASS у ресурсному файлі, цей параметр має бути встановлений у значення DLGWINDOWEXTRA .
hInstance	Ідентифікує екземпляр програми, для якого задається віконна функція.
hIcon	Ідентифікує клас піктограми для вікна. Може бути NULL . Константа IDI_APPLICATION оголошена у модулі Windows.pas і представляє собою ідентифікатор стандартної піктограми для програм.
hCursor	Ідентифікує клас курсора миші для вікна. Може бути NULL . Константа IDC_ARROW оголошена у модулі Windows.pas і представляє собою ідентифікатор стандартного курсора у виді стрілки.
hbrBackGround	Задає вид пензля, яким замальовуватиметься вікно. Змінна hbrBackGround містить значення, яке задається у аплеті "Екран" Панелі керування та задає стандартний колір для вікон операційної системи.
lpszMenuName	Вказує на рядок, який містить назву класу меню для вікна. Може бути NULL .
lpszClassName	Задає назву класу вікна.

Вказівник **lpfnWndProc** задає функцію, яка отримуватиме повідомлення, що посилатимуться до вікна, та при необхідності обробляти їх. Йому присвоюється адреса точки входу у функцію **WindowProc**⁵:

```
lpfnWndProc:=@WindowProc;
```

³ Всі використані функції **Windows API** описані у додатку. Для повного опису функцій **Windows API** див. довідку **MS SDK**.

⁴ У стандарті мови **Object Pascal** цей тип даних називається "запис" (record).

⁵ Оператор **@** вказує на адресу програмного об'єкта (змінної, функції, процедури...), перед яким стоїть.

Таким чином, всі повідомлення, пов'язані із створюваним вікном (рух миші, натискання її кнопки або клавіші на клавіатурі...), будуть передаватися у функцію вікна **WindowProc**. Прототип цієї функції жорстко визначений **Windows**: вона повинна отримувати чотири параметри певного типу – ідентифікатор вікна, повідомлення від якого вона приймає, саме повідомлення, та два параметри повідомлення.

Після заповнення структури **WindowClass** даними про клас вікна вона передається у функцію **RegisterClass**, яка реєструє його в операційній системі. Ця функція повертає атом⁶, який унікально ідентифікує створюваний клас. Якщо звертання до функції виявилось неуспішним, вона повертає 0, так що виклик можна вважати успішним, якщо результат не дорівнює нулю.

Після реєстрації класу вікна наступний крок – це створення вікна. У розглядуваній програмі цей процес здійснюється за допомогою функції **WinCreate**:

```
function WinCreate: HWND;
var hWindow : HWND;
begin
  hWindow:=CreateWindow(AppName, 'Вікно, створене з використанням API',WS_OVERLAPPEDWINDOW,
    100, 100, 600, 400, 0, 0, HInstance, nil);
  if hWindow<>0 then
  begin
    ShowWindow(hWindow, SW_SHOWNORMAL);
    UpdateWindow(hWindow);
    hBtnExit:=CreateWindow('BUTTON', 'Вихід',WS_CHILD or BS_DEFPUSHBUTTON or WS_TABSTOP, 500,
      10, 90, 30, hWindow, 0, HInstance, nil);
    if hBtnExit<>0 then ShowWindow(hBtnExit, SW_SHOWNORMAL);
    SendMessage(hBtnExit,WM_SETFONT, CreateFont(18,0,0,0,700,0,0,0,
      ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH,
      'Times New Roman Cyr'),1);
    hBtnAbout:=CreateWindow('BUTTON', 'Про вікно...',WS_CHILD or WS_TABSTOP, 500, 75, 90, 30,
      hWindow, 0, HInstance, nil);
    if hBtnAbout<>0 then ShowWindow(hBtnAbout, SW_SHOWNORMAL);
  end;
  Result:=hWindow;
end;
```

Ядром даної функції є звертання до функції **CreateWindow**, яка і здійснює створення вікна за шаблоном попередньо зареєстрованого класу. Перший параметр, який передається даній функції, представляє собою назву зареєстрованого класу. У даному випадку вікно, яке створюється, належить класу, назва якого міститься у константі **AppName='API Window'**, проте існує багато стандартних супутніх класів – кнопки, комбіновані списки, поля редагування, вікна списків і т.д., для створення яких не потрібно попередньо реєструвати їх клас.

Другим параметром функції **CreateWindow** є заголовок вікна, третій параметр вказує на стиль вікна і наступні чотири параметри визначають положення та розміри вікна. Можна також створити вікно стандартних розмірів та положення, при цьому у ролі цих чотирьох параметрів слід передати константу **CW_USEDEFAULT**. Наступні три параметри вказують відповідно на батьківське вікно (вікно-власник даного, новостворюваного вікна), на меню вікна, та на екземпляр прикладної програми, якому належить вікно. Оскільки в даному випадку створюється головне вікно програми, то у нього немає батьківського вікна. Меню у даному вікні також відсутнє, тому в ролі відповідного параметру передається нуль. Параметр **hInstance** доступний як глобальна змінна у проекті і вказує на даний екземпляр програми. Останній параметр можна заповнити визначеними користувачем даними, а також він грає певну роль у створенні вікон **MDI**. Проте в даному випадку він ігнорується (встановлений у **nil**).

Якщо виклик функції **CreateWindow** пройшов успішно, вона повертає ідентифікатор новоствореного вікна. В іншому випадку функція повертає нульове значення. Результату функції **WinCreate** присвоюється значення, яке повертає функція **CreateWindow**. Успішний виклик функції **CreateWindow** є необхідним для подальшої роботи програми (так як без головного вікна програма не

⁶ В **Microsoft® Windows®**, таблиця атомів – це системна таблиця, яка містить рядки і відповідні їм ідентифікатори. Програма поміщає рядок у таблицю атомів і отримує 16-бітне ціле число, яке називається атомом і може використовуватися для доступу до рядка. Рядок, який записується у таблицю атомів, називається назвою атому.

матиме функціональності), тому у випадку невдалого виклику цієї функції здійснюється вихід з програми.

Слід також відмітити, що при вдалому виклику функція **CreateWindow** посилає новоствореному вікну повідомлення **WM_CREATE**. Це повідомлення часто використовують для ініціалізації пов'язаних з вікном даних.

Після створення вікна викликаються дві функції – **ShowWindow** та **UpdateWindow**, які відповідно показують вікно зверху інших існуючих вікон та коректно перемальовують вікно.

Далі у програмі створюються дві кнопки – для виходу з програми ('**Вихід**') та для показу довідкової інформації про програму ('**Про вікно...**'). Створення та ініціалізація кнопки '**Вихід**' здійснюється таким чином:

```
hBtnExit:=CreateWindow('BUTTON', 'Вихід',WS_CHILD or BS_DEFPUSHBUTTON or WS_TABSTOP,
    500, 10, 90, 30, hWindow, 0, HInstance, nil);
if hBtnExit<>0 then ShowWindow(hBtnExit, SW_SHOWNORMAL);
SendMessage(hBtnExit,WM_SETFONT, CreateFont(18,0,0,0,700,0,0,0,ANSI_CHARSET,
    OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH,
    'Times New Roman Cyr'),1);
```

Саме створення кнопки здійснюється раніше описаною функцією **CreateWindow**, при цьому як назва класу вказується '**BUTTON**'. Так як вікно кнопки є потомком головного вікна програми, то відповідний параметр (**hWndParent**) при виклику вказує на ідентифікатор головного вікна програми **hWindow**, яке було створене раніше. Як стилі кнопки задано **WS_CHILD** (вікно є потомком вказаного), **BS_DEFPUSHBUTTON** (кнопка має вид кнопки, що використовується по замовчуванню), та **WS_TABSTOP** (при навігації по елементах керування вікна за допомогою клавіші **Tab** буде здійснюватись зупинка і на даній кнопці). Змінній **hBtnExit** присвоюється значення ідентифікатора вікна новоствореної кнопки. Якщо створення вікна кнопки було вдалим, воно показується за допомогою функції **ShowWindow**. Далі, для прикладу, проводиться зміна шрифту, яким пишеться напис на ній. Це робиться шляхом посилання вікну кнопки повідомлення **WM_SETFONT**:

```
SendMessage(hBtnExit,WM_SETFONT, CreateFont(18,0,0,0,700,0,0,0,ANSI_CHARSET,
    OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH,
    'Times New Roman Cyr'),1);
```

Перший параметр функції **SendMessage** – це ідентифікатор вікна, якому посилається повідомлення (у нашому випадку – **hBtnExit**), далі йде ідентифікатор повідомлення (**WM_SETFONT**). Перший параметр повідомлення повинен містити ідентифікатор логічного шрифту, тому в ролі цього параметра використано функцію **CreateFont**, яка створює логічний шрифт із заданими параметрами, та повертає його ідентифікатор. Другий параметр повідомлення може приймати значення 1 (це приведе до того, що вікно кнопки перемалює себе), або 0 (вікно не буде перемальовуватись після зміни шрифту). Так, в даному випадку, створюється логічний шрифт на основі шрифту 'Times New Roman Cyr', він має розмір 18 логічних одиниць та є жирним.

Створення кнопки '**Про вікно...**' здійснюється повністю аналогічно, проте для неї шрифт залишається без змін, стандартним.

Тепер залишилось створити найбільш динамічну частину програми – функцію вікна, яка прийматиме повідомлення та оброблятиме їх, забезпечуючи, зокрема, реакцію програми на дії користувача. Прототип віконної функції наступний⁷:

⁷ Оголошення та прототипи функцій **API** у даному документі приведено за стандартом, прийнятим у мові програмування **C++**. Згідно цього стандарту, функції оголошуються за такою схемою:

тип_функції назва_функції (список_параметрів)

При цьому список параметрів формується таким чином:

тип_параметру_1 назва_параметру_1, тип_параметру_2 назва_параметру_2, тип_параметру_3 назва_параметру_3...

```

LRESULT CALLBACK WindowProc(
    HWND hwnd,           // ідентифікатор вікна
    UINT uMsg,           // ідентифікатор повідомлення
    WPARAM wParam,       // перший параметр повідомлення
    LPARAM lParam        // другий параметр повідомлення
);

```

Першим її параметром є ідентифікатор вікна, повідомлення від якого приймає функція, далі слідує ідентифікатор прийнятого повідомлення та два його параметри, які містять дані, що залежать від типу повідомлення. Текст віконної функції для розглядуваного прикладу наступний:

```

function WindowProc(Window:HWND; AMessage, WPARAM, LPARAM:longint):longint; stdcall; export;
begin
    WindowProc:=0;
    case AMessage of
        WM_DESTROY: begin PostQuitMessage(0); Exit; end;
        WM_COMMAND: if LParam=hBtnExit then begin PostQuitMessage(0); Exit; end
                     else if LParam=hBtnAbout then MessageBox(Window,
                        'Вікно, створене функціями API, без використання VCL Delphi.'+#13#13+
                        'Copyright© Microsoft... і т.д.', 'API Window', MB_OK or MB_ICONINFORMATION);
        WM_KEYDOWN: if (WParam=VK_RETURN) or (WParam=VK_ESCAPE) then
                        SendMessage(hBtnExit, BM_CLICK, 0, 0)
                     else if WParam=VK_F1 then SendMessage(hBtnAbout, BM_CLICK, 0, 0);
    end;
    WindowProc:=DefWindowProc(Window, AMessage, WPARAM, LPARAM);
end;

```

Дана функція обробляє три повідомлення:

- **WM_DESTROY** – виникає, коли вікно знищується та посилається віконній функції після того, як воно забирається з екрану. Це повідомлення не має параметрів.
- **WM_COMMAND** – виникає, коли користувач вибирає будь-який елемент керування (кнопку, елемент меню, клавішу-акселератор...). При цьому значення вищого слова першого параметра повідомлення (**WParam**) містить код повідомлення, якщо воно надходить від елемента керування вікна. Якщо повідомлення від акселератора, цей параметр дорівнює 1, якщо від елемента меню – 0. Значення нижнього слова цього параметра містить код елемента керування, від якого отримано повідомлення. Другий параметр (**LParam**) містить ідентифікатор (**handle**) елемента керування, від якого отримано повідомлення.
- **WM_KEYDOWN** – виникає, якщо користувач натискає будь-яку клавішу на клавіатурі. При цьому перший параметр повідомлення містить віртуальний код клавіші, яка була натиснена, а другий – дані про стан клавіатури (кількість повторів, скан-код клавіші...).

Оскільки у даній програмі всього одне вікно, то, якщо отримується повідомлення **WM_DESTROY**, слід припинити роботу всієї програми. Це здійснюється за допомогою функції **PostQuitMessage(0)**, яка заставляє **Windows** послати у чергу повідомлень вікна повідомлення **WM_QUIT**, що означає вказівку програмі завершити свою роботу. При цьому здійснюється вихід із циклу обробки повідомлень, оскільки функція **GetMessage** повертає нульове значення (**false**) лише при отриманні нею повідомлення **WM_QUIT**:

```

while GetMessage(AMessage, 0, 0, 0) do
begin
    TranslateMessage(AMessage);
    DispatchMessage(AMessage);
end;

```

Повідомлення **WM_COMMAND** використовується для програмування реакції на натискання тих чи інших кнопок у програмі. При обробці цього повідомлення здійснюється перевірка значення параметра **LParam**, так як він містить ідентифікатор елемента керування, від якого надійшло повідомлення. Якщо цей ідентифікатор рівен **hBtnExit** (тобто натиснено на кнопку "Вихід", оскільки у змінну **hBtnExit** після створення цієї кнопки поміщується ідентифікатор її вікна), то здійснюється вихід з програми через виклик функції **PostQuitMessage**. Якщо ж параметр **LParam** рівен **hBtnAbout** (тобто натиснено кнопку "Про вікно..."), то за допомогою функції **MessageBox** здійснюється показ

стандартного вікна з повідомленням про дану програму. При цьому параметр **MB_OK** вказує, що у вікні повідомлення є одна кнопка – "ОК", а параметр **MB_ICONINFORMATION** – що воно буде мати стиль інформативного вікна (у ньому буде намальовано синю піктограму інформативності з літерою "i").

І останнє повідомлення, яке обробляється описуваною віконною функцією – це повідомлення **WM_KEYDOWN**, яке виникає при натисканні на клавіші клавіатури. При надходженні цього повідомлення параметр **WParam** містить віртуальний код натисненої клавіші. Тому, аналізуючи цей параметр, можна визначати, яка клавіша клавіатури була натиснена. У даній програмі, при натисканні клавіш "Enter" або "Esc" (їх віртуальні коди відповідно рівні **VK_RETURN** та **VK_ESCAPE**) здійснюється емуляція натискання на кнопку вікна "Вихід", а при натисканні на клавішу "F1" (її віртуальний код рівен **VK_F1**) емується натискання клавіші "Про вікно...". Емуляція здійснюється шляхом посилання вікну відповідної кнопки повідомлення **BM_CLICK**. Так, для посилання цього повідомлення вікну кнопки "Вихід", код має виглядати так:

```
SendMessage(hBtnExit, BM_CLICK, 0, 0)
```

Дане повідомлення не має параметрів, тому їх значення встановлено в нулі.

Виклик функції **DefWindowProc** в кінці віконної процедури дозволяє операційній системі здійснити обробку стандартних повідомлень самостійно. Так, всі вікна "знають", як обробляти повідомлення перемальовування, переміщення та зміни розмірів вікна. Для того, щоб автоматично гарантувати таку поведінку, слід передати всі необроблені повідомлення функції **DefWindowProc**. Цю функцію можна вважати своєрідним продовженням вже розглянутої віконної функції

3. Завдання на лабораторну роботу

Написати (з використанням лише функцій **Windows API**, без впровадження компонентів **VCL Delphi**) програму, яка створюватиме вікно, що міститиме елементи керування, вказані у таблиці №1.2. Крім цього, програма повинна містити кнопку "Вихід", за якою буде здійснюватись коректне завершення роботи програми, та кнопку "Автор", за якою повинно виводитись вікно повідомлення про автора програми та групу, у якій він (вона) навчається.

Таблиця №1.2.

Варіант	Завдання
1	Три елемента статичного тексту, два елемента редагування та один комбінований список.
2	Дві кнопки, два комбінованих списки, та два списки.
3	Чотири елемента текстового вводу, кнопка та комбінований список.
4	Два елемента статичного тексту, два поля редагування, два списки.
5	Елемент статичного тексту, чотири комбінованих списки та один звичайний список.
6	Елемент редагування, три елемента статичного тексту, кнопка та комбінований список.
7	Два елемента статичного тексту, елемент редагування та три кнопки.
8	Три елемента статичного тексту, два поля вводу та кнопка.
9	Три елемента текстового вводу, кнопка та два комбінованих списки.
10	Елемент статичного тексту, чотири списки та одне поле редагування.
11	Дві кнопки, два поля редагування, та два списки.
12	Два елемента статичного тексту, елемент списку та три кнопки.
13	Три елемента статичного тексту, два поля вводу та комбінований список.
14	Два елемента редагування, елемент списку та три комбінованих списки.
15	Три комбінованих списки, два звичайних списки, та поле редагування.
16	Два елемента статичного тексту, два поля редагування та два комбінованих списки.
17	Чотири кнопки, список та комбінований список.
18	Три елемента текстового вводу, кнопка та два статичних тексти.
19	Два елемента статичного тексту, два елемента списку та два комбінованих списки.
20	Елемент редагування, елемент статичного тексту, три кнопки та комбінований список.
21	Дві кнопки, два списки та два комбінованих списки.
22	Статичний текст, список, два комбінованих списки та поле редагування.
23	Два поля редагування, два списки, кнопка та комбінований список.
24	Два елемента статичного тексту, два поля редагування, та дві кнопки.

Варіант	Завдання
25	Три списки, два комбінованих списки, кнопка та статичний текст.
26	Три комбінованих списки, елемент редагування та дві кнопки.
27	Дві кнопки, елемент статичного тексту, і чотири комбінованих списки.
28	Елемент редагування, дві кнопки, список і два комбінованих списки.
29	Три поля редагування, дві кнопки, список і статичний текст.
30	Три елементи статичного тексту, поле редагування, кнопка і список.

4. Текст програми

```

program Window;

uses Windows, Messages;

const AppName='API Window';

var AMessage : TMsg;
    hWindow   : HWND;
    hBtnExit  : HWND;
    hBtnAbout : HWND;

function WindowProc(Window:HWND; AMessage, WParam, LParam:longint):longint; stdcall; export;
begin
    WindowProc:=0;
    case AMessage of
        WM_DESTROY: begin PostQuitMessage(0); Exit; end;
        WM_COMMAND: if LParam=hBtnExit then begin PostQuitMessage(0); Exit; end
                     else if LParam=hBtnAbout then MessageBox(Window,
                        'Вікно, створене функціями API, без використання VCL Delphi.'+#13#13+
                        'Copyright© Microsoft... і т.д.', 'API Window', MB_OK or MB_ICONINFORMATION);
        WM_KEYDOWN: if (WParam=VK_RETURN) or (WParam=VK_ESCAPE) then
                        SendMessage(hBtnExit, BM_CLICK, 0, 0)
                     else if WParam=VK_F1 then SendMessage(hBtnAbout, BM_CLICK, 0, 0);
    end;
    WindowProc:=DefWindowProc(Window, AMessage, WParam, LParam);
end;

function WinRegister: boolean;
var WindowClass: TWndClass;
begin
    with WindowClass do begin
        Style:=CS_HREDRAW or CS_VREDRAW;
        lpfnWndProc:=@WindowProc;
        cbClsExtra:=0;
        cbWndExtra:=0;
        hInstance:=HInstance;
        hIcon:=LoadIcon(0, IDI_APPLICATION);
        hCursor:=LoadCursor(0, IDC_ARROW);
        hbrBackGround:=COLOR_WINDOW;
        lpszMenuName:=nil;
        lpszClassName:=AppName;
    end;
    Result:=RegisterClass(WindowClass)<>0;
end;

function WinCreate: HWND;
var hWindow : HWND;
begin
    hWindow:=CreateWindow(AppName, 'Вікно, створене з використанням API', WS_OVERLAPPEDWINDOW,
        100, 100, 600, 400, 0, 0, HInstance, nil);

    if hWindow<>0 then
    begin
        ShowWindow(hWindow, SW_SHOWNORMAL);
        UpdateWindow(hWindow);
        hBtnExit:=CreateWindow('BUTTON', 'Вихід', WS_CHILD or BS_DEFPUSHBUTTON or WS_TABSTOP, 500,
            10, 90, 30, hWindow, 0, HInstance, nil);
        if hBtnExit<>0 then ShowWindow(hBtnExit, SW_SHOWNORMAL);
    end;
end;

```

```
SendMessage(hBtnExit,WM_SETFONT, CreateFont(18,0,0,0,700,0,0,0,
ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH,
'Times New Roman Cyr'),1);
hBtnAbout:=CreateWindow('BUTTON', 'Про вікно...',WS_CHILD or WS_TABSTOP, 500, 75, 90, 30,
hWindow, 0, HInstance, nil);
if hBtnAbout<>0 then ShowWindow(hBtnAbout, SW_SHOWNORMAL);
end;
Result:=hWindow;
end;

begin
if not WinRegister then
begin
  MessageBox(0,'Клас вікна не зареєстровано',AppName,MB_OK);
  Exit;
end;
hWindow:=WinCreate;
if hWindow=0 then
begin
  MessageBox(0,'Не вийшло створити вікно.',AppName,MB_OK);
  Exit;
end;
while GetMessage(AMessage,0,0,0) do
begin
  TranslateMessage(AMessage);
  DispatchMessage(AMessage);
end;
Halt(AMessage.wParam);
end.
```


Лабораторна робота №2

Тема: Створення діалогових вікон засобами Windows API та їх використання у прикладних програмах.

Мета: Вивчення основних принципів створення діалогових вікон у прикладних програмах за допомогою мови сценарії ресурсів (Resource Script Language). Вивчення методів взаємодії діалогового вікна з потоком-власником та операційною системою через систему повідомлень.

1. Створення діалогового вікна прикладної програми засобами Windows API

У пункті 3 методичних вказівок приведено текст програми, головне вікно якої (рис. 2.1) містить три кнопки – "Вихід" (для завершення роботи із програмою), "Діалог" (для показу діалогового вікна), та "Про вікно..." (для виводу вікна з коротким повідомленням про програму).

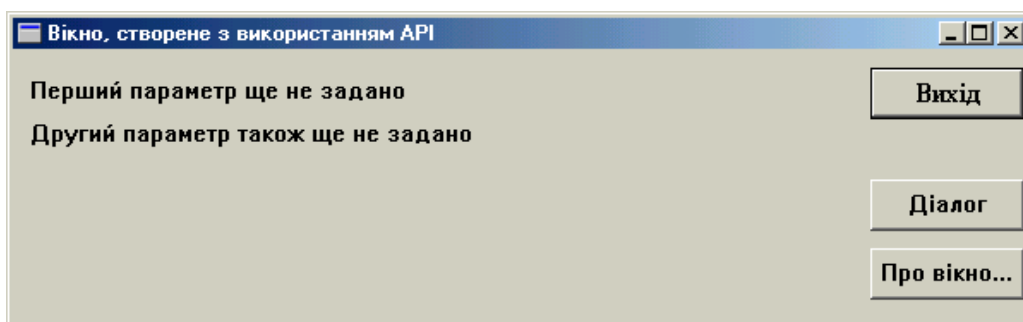


Рис. 2.1. Головне вікно програми

У вікні містяться два елементи статичного тексту (API-аналог компонента **TLabel** з **VCL Delphi**), які можуть відображувати значення двох параметрів, якщо вони будуть задані за допомогою діалогового вікна (кнопка "Діалог"). Вигляд діалогового вікна показано на рис. 2.2.

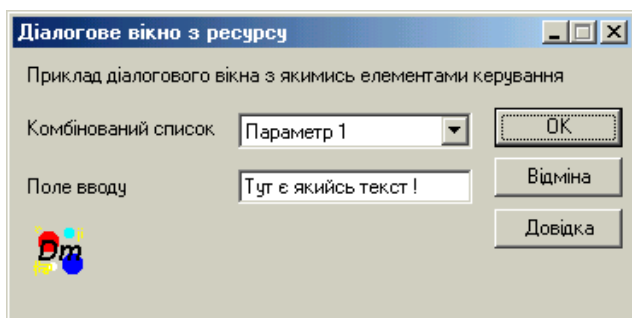


Рис. 2.2. Діалогове вікно

Воно дає можливість задавати два параметри- один з них вибирається з комбінованого списку (і може приймати значення "Параметр 1", "Параметр 2"...), а другий параметр є текстовим і задається з клавіатури у полі текстового вводу. Крім цього, у діалоговому вікні в нижньому лівому куті розташовано піктограму програми.

Для створення шаблону діалогового вікна використовують мову сценаріїв ресурсів (**Resource Script Language**), за допомогою якої описують вид вікна та інші ресурси, які у ньому використовуватимуться (малюнки, меню та інші). Так, вид сценарію для показаного на рис. 2.2 діалогового вікна наступний:

```
#define IDC_COMBOBOX1 101
#define IDC_EDIT1 102
ICIcon ICON "demo.ico"
ICDialog DIALOG 70, 50, 241, 96
STYLE DS_SYSMODAL | DS_MODALFRAME | WS_OVERLAPPED | WS_VISIBLE | WS_CAPTION | WS_SYSMENU |
WS_THICKFRAME | WS_MINIMIZEBOX
CAPTION "Діалогове вікно з ресурсу"
FONT 8, "MS Sans Serif"
{
```

```

DEFPUSHBUTTON "ОК", IDOK, 188, 21, 50, 14
PUSHBUTTON "Відміна", IDCANCEL, 188, 39, 50, 14
PUSHBUTTON "Довідка", IDHELP, 188, 57, 50, 14
COMBOBOX IDC_COMBOBOX1, 88, 23, 91, 33, CBS_DROPDOWNLIST | WS_VSCROLL | WS_GROUP | WS_TABSTOP
LTEXT "Приклад діалогового вікна з якимись елементами керування", 10, 5, 5, 240, 8
EDITTEXT IDC_EDIT1, 88, 43, 91, 12
LTEXT "Комбінований список", 11, 5, 24, 79, 8
LTEXT "Поле вводу", 12, 5, 45, 60, 8
ICON "ICIcon", 20, 7, 62, 18, 20
}

```

Призначення основних елементів синтаксису мови сценаріїв ресурсів приведено у таблиці №2.1.

Таблиця №2.1

Директива, команда	Синтаксис	Призначення
#define	#define identifier text	При компіляції значення identifier розпізнається компілятором і автоматично замінюється на значення text Приклад: #define IDC_COMBOBOX1 101
ICON , оголошення типу 1	resource-name ICON filename	Створює ресурс піктограми з назвою resource-name на основі заданого файлу filename Приклад: ICIcon ICON "demo.ico"
ICON , оголошення типу 2	ICON resource-name, control-ID, x, y, width, height	Розміщує у вікні піктограму, попередньо оголошену згідно типу 1, у заданому положенні (x та y) та заданих розмірів (width, height). Піктограмі присвоюється ідентифікатор control-ID. Приклад: ICON "ICIcon", 20, 7, 62, 18, 20
DIALOG	resource-name DIALOG x, y, width, height [STYLE w-style] [CAPTION w-cap] [MENU res-name] [CLASS w-class] [FONT f-spec] BEGIN dialog-controls END	Описує діалогове вікно. Може включати стиль вікна, його клас, розмір (width, height), положення (x та y), та елементи керування (які описуються в межах конструкції BEGIN... END [або {... }]).
STYLE	STYLE w-style	Задає константи стилю діалогового вікна (w-style), які поєднуються оператором or (або). DS_SYSMODAL - створює системне модальне діалогове вікно DS_MODALFRAME - в поєднанні з WS_CAPTION створює рухоме вікно із сталими розмірами WS_OVERLAPPED - створює вікно верхнього рівня WS_VISIBLE - при створенні вікно є видимим WS_CAPTION - вікно має полосу заголовка WS_SYSMENU - вікно містить системне меню в заголовку WS_THICKFRAME - вікно має тонку рамку WS_MINIMIZEBOX - вікно містить кнопку мінімізації
CAPTION	CAPTION w-cap	Задає заголовок w-cap для діалогового вікна.
MENU	MENU res-name	Задає ресурс меню діалогового вікна, який повинен бути попередньо описаним.
CLASS	CLASS w-class	Задає клас діалогового вікна. Якщо конструкція відсутня, створюється стандартне вікно.
FONT	FONT f-spec	Задає шрифт для діалогового вікна. f-spec задає розмір та назву шрифта. Приклад: FONT 8, "MS Sans Serif"
BEGIN END або { }		Оператори задають початок та кінець єдиної багатолінійної конструкції.

Директива, команда	Синтаксис	Призначення
DEFPUSHBUTTON	DEFPUSHBUTTON text, control-ID, x, y, width, height, [c-style]	Створює кнопку по замовчуванню у діалоговому вікні. Кнопка містить напис text, ідентифікатор ресурсу control-ID, екранні координати x, y, та розміри width, height. Кнопка може містити стилі: WS_DISABLED - елемент керування неактивний WS_GROUP - елемент згрупований WS_TABSTOP - при навігації по елементам керування вікна за допомогою клавіші Tab здійснюватиметься зупинка на даній кнопці. Приклад: DEFPUSHBUTTON "ОК", IDOK, 188, 21, 50, 14
PUSHBUTTON	PUSHBUTTON text, control-ID, x, y, width, height, [c-style]	Створює кнопку у діалоговому вікні. Кнопка містить напис text, ідентифікатор ресурсу control-ID, екранні координати x, y, та розміри width, height. Кнопка може містити стилі [c-style]. Приклад: PUSHBUTTON "Відміна", IDCANCEL, 188, 39, 50, 14
COMBOBOX	COMBOBOX ID, x, y, width, height, [c-style]	Створює комбінований список у діалоговому вікні. Він містить ідентифікатор ресурсу ID, екранні координати x, y, розміри width, height, та може містити стилі [c-style]. Приклад: COMBOBOX 2, 88, 23, 91, 33, CBS_DROPDOWNLIST WS_VSCROLL WS_GROUP WS_TABSTOP
LTEXT	LTEXT text, control-ID, x, y, width, height, [c-style]	Створює вирівняний по лівому краю текст text у діалоговому вікні. Він містить ідентифікатор ресурсу control-ID, екранні координати x, y, розміри width, height, та може містити стилі [c-style]. Приклад: LTEXT "Текст", 12, 5, 45, 60, 8
EDITTEXT	EDITTEXT control-ID, x, y, width, height, [c-style]	Створює поле текстового вводу (поле редагування) у діалоговому вікні. Воно містить ідентифікатор ресурсу control-ID, екранні координати x, y, розміри width, height, та може містити стилі [c-style]. Приклад: EDITTEXT 1, 88, 43, 91, 12

Файл сценарію зберігається як звичайний текстовий файл ⁸, і компілюється, наприклад, з використанням компілятора ресурсів фірми Borland BRCC32.EXE, який поставляється разом з Delphi⁹. Якщо файл сценарію назвати **ICDIALOG.RC**, то в результаті його компіляції отримається файл ресурсів **ICDIALOG.RES**, який міститиме шаблон розробленого діалогу. Для під'єднання цього ресурсного файлу до проекту Delphi використовується така директива компілятора ¹⁰:

```
{ $R ICDIALOG.RES }
```

Якщо шаблон діалогового вікна вже створено та відповідний ресурсний файл під'єднано до проекту, то само вікно можна створити за допомогою функції **DialogBox** ¹¹:

```
DialogBox(hInstance, 'ICDialog', hWindow, @DlgProc)
```

Ця функція як параметри отримує ідентифікатор екземпляра запущеної програми **hInstance**, назву ресурсу діалогового вікна (в даному випадку **'ICDialog'** – див. описаний вище сценарій діалогового вікна), ідентифікатор вікна-власника для створюваного діалогу (**hWindow**), та адресу

⁸ Для цього можна використати стандартний текстовий редактор "Блокнот" (Notepad) Windows.

⁹ При цьому назва файлу сценарію, який слід скомпілювати, вказується як перший параметр командного рядка компілятора. Так, якщо файл сценарію має назву **ICDIALOG.RC**, то командний рядок має виглядати так:
BRCC32.EXE ICDIALOG.RC

¹⁰ Її можна помістити відразу після директиви, яку Delphi додає у проект автоматично – { \$R *.res }, і яка під'єднує файл ресурсів самого проекту (його назва співпадає з назвою проекту).

¹¹ Опис функцій **Windows API** приведено у додатку. Для повного опису функцій Windows API див. довідку MS SDK.

точки входу віконної функції діалогового вікна (@DlgProc¹²). Як результат функція **DialogBox** повертає ідентифікатор натисненої кнопки.

Нам необхідно, щоб при натисканні кнопки "ОК" у діалоговому вікні задані у ньому два параметри зберігалися у глобальних змінних. В описуваній програмі такими змінними є: **ParamEdit** – для зберігання тексту з поля редагування, та **ParamCB** – для зберігання тексту з комбінованого списку. У **Windows API** для зберігання текстової інформації використовуються рядки з кінцевим нулем (у **Object Pascal** стандартним засобом для цього є тип **string**). Рядок з кінцевим нульовим символом в **Object Pascal** оголошується як тип **PChar**, що представляє собою вказівник¹³ на буфер, у якому розміщено рядок. Оскільки рядок представляє собою масив символів (який закінчується символом з кодом 0), то його можна й оголосити як масив (**array**).

Для ілюстрації різних методів роботи з рядками, які містять кінцевий нуль, у даній програмі змінні двох параметрів оголошені по-різному:

```
ParamEdit : array [1..50] of char;
ParamCB   : PChar;
```

При оголошенні змінної як масив у сегменті даних програми резервується відповідне місце для елементів цього масиву, що приводить до збільшення розмірів виконавчого файлу на розмір масиву. Якщо змінна оголошується як тип **PChar** (тобто вказівник, який без попередньої ініціалізації може вказувати на довільну ділянку пам'яті), то перед використанням відповідної змінної слід виділити необхідний обсяг пам'яті для рядка:

```
GetMem(ParamCB,50);
```

В даному випадку виділяється 50 байт, що не дозволить обробляти рядки довжиною більше 49 символів¹⁴. Так як у програмі не здійснюється обмеження по довжині введеного тексту та відповідна перевірка, то при спробі роботи з довшими рядками може виникати помилка доступу до пам'яті. Після того, як буфер для рядка вже непотрібний, слід вивільнити попередньо зарезервовану пам'ять:

```
FreeMem(ParamCB);
```

Розглянемо віконну функцію для створеного діалогового вікна:

```
function DlgProc(Window : hWnd; Msg,WParam,LParam : Integer): Integer; stdcall;
begin
  Result:=0;
  case Msg of
    WM_INITDIALOG : begin
      // Ініціалізація діалогового вікна
      Result:=0;
      hcbType:=GetDlgItem(Window,101);
      SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 1')));
      SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 2')));
      SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 3')));
      SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 4')));
      SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 5')));
      SendMessage(hcbType,CB_SETCURSEL,0,0);

      hEdit:=GetDlgItem(Window,102);
      SendMessage(hEdit,WM_SETTEXT,0,DWORD(PChar('Тут є якийсь текст !')));
    end;
    WM_COMMAND : if (LoWord(WParam)=IDOK) then begin // Натиснено кнопку "ОК"
      SendMessage(hEdit,WM_GETTEXT,
        SendMessage(hEdit,WM_GETTEXTLENGTH,0,0)+1,DWORD(@ParamEdit));
    end;
  end;
end;
```

¹² Оператор @ вказує на адресу програмного об'єкта (змінної, функції, процедури...), перед яким стоїть.

¹³ Вказівник – це змінна, яка містить адресу програмного об'єкта (тобто іншої змінної, масиву, структури, процедури, функції і т.д.)

¹⁴ Один байт має залишатись для кінцевого символа з кодом 0.

```

SendMessage(hcbType,WM_GETTEXT,
    SendMessage(hcbType,WM_GETTEXTLENGTH,0,0)+1,DWORD(ParamCB));
EndDialog(Window,idOK)
end
// Натиснено кнопку "Відміна"
else if (LoWord(WParam)=IDCANCEL) then EndDialog(Window,idCancel)
// Натиснено кнопку "Довідка"
else
    if (LoWord(WParam)=IDHELP) then MessageBox(Window,
        'Довідка про діалогове вікно, яке створене з використанням '+
        'Microsoft® Windows® Application Program Interface®.'+
        '#13#13+\"Copyright© Microsoft® & К°',
        'Довідка про API Window',MB_OK or MB_ICONINFORMATION);
WM_CLOSE      : EndDialog(Window,idCancel);
else Result:=0;
end;
end;
end;

```

Повідомлення **WM_INITDIALOG** використовується для ініціалізації новоствореного діалогового вікна, оскільки воно отримується віконною функцією один раз безпосередньо після його створення. Так як для роботи з будь-яким вікном (в тому числі з вікном елемента керування) потрібно знати його ідентифікатор, то спочатку отримуємо ідентифікатор комбінованого списку:

```
hcbType:=GetDlgItem(Window,101);
```

Функція **GetDlgItem** повертає ідентифікатор вікна елемента керування діалогового вікна за його номером. У даному випадку визначається ідентифікатор комбінованого списку (його номер у шаблоні діалогового вікна – 101 (див. приведений вище текст сценарію діалогового вікна)). Після цього у комбінований список додаються рядки з необхідними значеннями. Це здійснюється шляхом посилання його вікна повідомлення **CB_ADDSTRING**:

```
SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 1')));
```

Другий параметр цього повідомлення містить адресу рядка 'Параметр 1', який і добавиться у список. Таким чином у список поміщуються п'ять рядків з різними значеннями. Після цього визначається ідентифікатор поля текстового вводу **hEdit**:

```
hEdit:=GetDlgItem(Window,102);
```

За допомогою повідомлення **WM_SETTEXT** у це поле вноситься текст ('Тут є якийсь текст !'):

```
SendMessage(hEdit,WM_SETTEXT,0,DWORD(PChar('Тут є якийсь текст !')));
```

На цьому ініціалізація вікна закінчується.

Повідомлення **WM_COMMAND** в нашому випадку отримується при натисканні на кнопки діалогового вікна. При цьому нижнє слово параметра **wParam** містить ідентифікатор ресурсу елемента керування, від якого отримано повідомлення (див. текст сценарію діалогового вікна). Так, наприклад, для кнопки "ОК" для описуваного діалогу цей ідентифікатор дорівнює **IDOK**. Якщо у діалоговому вікні було натиснено кнопку "ОК", то текст з комбінованого списку та поля редагування зчитується і запам'ятовується у відповідних змінних.

Для отримання тексту з відповідного текстового вікна йому посилається повідомлення **WM_GETTEXT**. При цьому перший параметр повідомлення має вказувати кількість символів, які будуть отримані з вікна, а другий параметр – на буфер, що отримає значення рядка. Для обчислення довжини рядка у вікні йому посилається повідомлення **WM_GETTEXTLENGTH**, яке результатом повертає кількість символів у рядку вікна. Так як при цьому не враховується останній, нульовий символ, то для задання розміру тексту до результату повідомлення **WM_GETTEXTLENGTH** додається 1, і результат цієї операції передається як перший параметр повідомлення **WM_GETTEXT**. Другий параметр цього повідомлення повинен містити адресу рядка, який прийме текстове значення. При цьому для змінної **ParamEdit**, яка оголошена як масив символів, визначається адреса першого елемента масиву (**@ParamEdit**) та приводиться до типу **DWORD**, щоб бути сумісним з відповідним параметром повідомлення:

```
SendMessage(hEdit,WM_GETTEXT,SendMessage(hEdit,WM_GETTEXTLENGTH,0,0)+1,DWORD(@ParamEdit));
```

Для змінної **ParamCB** (типу **PChar**), яка сама є вказівником на рядок, адресу не потрібно обчислювати:

```
SendMessage(hcbType,WM_GETTEXT,SendMessage(hcbType,WM_GETTEXTLENGTH,0,0)+1,DWORD(ParamCB));
```

Після отримання значень, заданих у діалоговому вікні, воно знищується за допомогою функції **EndDialog**, яка повертає значення **IDOK**.

Якщо у діалоговому вікні було натиснено кнопку "**Відміна**", воно знищується, повертаючи результат **IDCANCEL**. При виборі кнопки "**Довідка**" з'являється вікно з повідомленням про дане діалогове вікно.

Останнє повідомлення, яке обробляється віконною функцією діалогового вікна- **WM_CLOSE**. При цьому воно також знищується.

Якщо у діалоговому вікні натискається кнопка "**ОК**", викликається процедура **ChangeParams** (див. віконну функцію головного вікна **WindowProc**), яка відображає вибрані в діалозі параметри у головному вікні програми за допомогою елементів статичного тексту:

```
procedure ChangeParams;
begin
  SendMessage(hText1,WM_SETTEXT,0,DWORD('Вибраний параметр з комбінованого списку: '+ParamCB));
  SendMessage(hText2,WM_SETTEXT,0,DWORD(@ParamEdit));
end;
```

Запис тексту у відповідний елемент здійснюється шляхом посилання йому повідомлення **WM_SETTEXT**, другий параметр якого містить адресу буфера, що містить рядок для запису у вікно. Таким чином введені у діалоговому вікні параметри стають доступними в головному вікні програми.

2. Завдання на лабораторну роботу

Написати (з використанням лише функцій **Windows API**, без впровадження компонентів **VCL Delphi**) програму, яка здійснюватиме розрахунок арифметичного виразу згідно варіанту (таблиця №2.2) та відображатиме його в головному вікні у полі редагування (**EDIT**). Значення змінних, які використовуються для розрахунку, повинні задаватися за допомогою окремого діалогового вікна. Програма повинна здійснювати перевірку можливості обчислення виразу при заданих параметрах та у випадку неможливості проведення обчислень видавати відповідне повідомлення¹⁵.

Головне вікно має містити кнопку "**Вихід**", за якою здійснюватиметься коректне завершення роботи програми.

Таблиця №2.2

Варіант	Завдання
1	$y = \cos(ab) \cdot \sqrt{ab}$
2	$y = \frac{\sqrt{a-b^3}}{\sin\left(ab + \frac{a}{b}\right)}$
3	$y = \cos^3(a) + \sin^2(b)$

¹⁵ Наприклад, квадратний корінь з від'ємного числа не може бути обчислено.

Варіант	Завдання
4	$y = \frac{\ln(a)}{a + 4.23b + ab}$
5	$y = (356a - 23b)(\sqrt{3a - b})$
6	$y = 2.36\pi + 2.58\cos^5(3ab)$
7	$y = 2a + \frac{3b^3 - 6a}{6a^2 - b}$
8	$y = 78a^6 + 32b^4 - 23ab^3$
9	$y = \frac{- a^3 + 4b^5 }{\sqrt[3]{3ab}}$
10	$y = \frac{\sqrt[4]{a + 3b}}{(2a + b)(3b - 2a)}$
11	$y = \frac{-\ln(3\pi a)}{\cos(2a + 3b^{1/3})}$
12	$y = \frac{\sqrt[5]{a + (3b - a^2)}}{4.36b + 0.23a}$
13	$y = \sqrt[5]{(3a + 6b)} + \frac{23a}{6b}$
14	$y = 156a + \sqrt[3]{3a + 4.25b} + \sqrt{0.23ab}$
15	$y = \frac{\lg(3.3a + 4.9b^2)}{\sqrt{3a}}$
16	$y = -4a - 6b^3 - 34ab - \frac{a}{b}$
17	$y = 5a + 45 \frac{b^2}{-4a^{1/3}} + 3ab$
18	$y = \frac{-2ab + \sqrt{ab}}{3a(4a + 3b^2)}$
19	$y = 2a + 3b^5 + \frac{6ab}{3a^2}$
20	$y = \frac{3a}{2b^2} + \frac{23ab}{0.69a^2} + \frac{\sqrt{ab}}{a}$
21	$y = \sqrt{\frac{x_2^2 + x_1 / x_2}{\cos(x_1^3 x_2^5) + 2x_1}}$
22	$y = \cos(\sqrt{x_2} + 34 \cdot \sin(x_1)) - 4\sin(x_2)$
23	$y = \frac{1}{4 + x_2} \cdot \sqrt{\cos^2\left(\lg_{10} \frac{x_2}{x_1}\right)}$
24	$y = \sqrt{56x_1 + \frac{x_1 + x_2 + \sin(x_1 x_2)}{5 - \cos(x_2^2)}}$

Варіант	Завдання
25	$y = \frac{3x_2 - x_1^2}{\cos^3\left(\lg_{10} \frac{x_1 + 2x_2 + 9}{0.37}\right)}$
26	$y = \frac{\ln(x_2)}{\sqrt[5]{0.6x_1 \sin x_2 \cos x_1^4}}$
27	$y = \frac{\sqrt{\cos^3(x_1) + x_2}}{x_1^{13} + 3/\cos(x_2)}$
28	$y = \frac{5\sqrt{x_1^3 + x_2^5 - \cos(x_2)}}{\operatorname{tg}(x_1)}$
29	$y = \sqrt{\frac{\cos(2x_2) + x_1/x_2}{16x_2x_1}}$
30	$y = \sum_{j=-10}^{100} 0.1x_1 \sin x_2 \cos x_1^j + 55^j$

3. Текст програми

```

program Window;

uses Windows, Messages, SysUtils;

const AppName='API Window';

var AMessage      : TMsg;
    hWindow       : HWnd;
    hBtnExit      : HWnd;
    hBtnAbout     : HWnd;
    hBtnDlg       : HWnd;
    hText1        : HWnd;
    hText2        : HWnd;

var hcbType : HWnd;
    hEdit   : HWnd;

    ParamEdit : array [1..50] of char;
    ParamCB   : PChar;

{$R ICDIALOG.RES}

function DlgProc(Window : HWnd; Msg,WParam,LParam : Integer): Integer; stdcall;
begin
    Result:=0;
    case Msg of
        WM_INITDIALOG : begin
            // Ініціалізація діалогового вікна
            Result:=0;
            hcbType:=GetDlgItem(Window,101);
            SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 1')));
            SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 2')));
            SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 3')));
            SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 4')));
            SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 5')));
            SendMessage(hcbType,CB_SETCURSEL,0,0);

            hEdit:=GetDlgItem(Window,102);
            SendMessage(hEdit,WM_SETTEXT,0,DWORD(PChar('Тут є якийсь текст !')));
        end
    end;
end;

```

```

end;
WM_COMMAND : if (LoWord(WParam)=IDOK) then begin // Натиснено кнопку "ОК"
    SendMessage(hEdit,WM_GETTEXT,
        SendMessage(hEdit,WM_GETTEXTLENGTH,0,0)+1,DWORD(@ParamEdit));
    SendMessage(hcbType,WM_GETTEXT,
        SendMessage(hcbType,WM_GETTEXTLENGTH,0,0)+1,DWORD(ParamCB));
    EndDialog(Window,idOK)
end
// Натиснено кнопку "Відміна"
else if (LoWord(WParam)=IDCANCEL) then EndDialog(Window,idCancel)
else // Натиснено кнопку "Довідка"
    if (LoWord(WParam)=IDHELP) then MessageBox(Window,
        'Довідка про діалогове вікно, яке створене з використанням Microsoft® '+
        'Windows® Application Program Interface®.'+
        #13#13+'Copyright© Microsoft® & К°',
        'Довідка про API Window',MB_OK or MB_ICONINFORMATION);
WM_CLOSE : EndDialog(Window,idCancel);
else Result:=0;
end;
end;

procedure ChangeParams;
begin
    SendMessage(hText1,WM_SETTEXT,0,DWORD('Вибраний параметр з комбінованого списку: '+ParamCB));
    SendMessage(hText2,WM_SETTEXT,0,DWORD(@ParamEdit));
end;

function WindowProc(Window:HWND; AMessage, WParam, LParam:longint):longint;stdcall; export;
begin
    WindowProc:=0;
    case AMessage of
        WM_DESTROY: begin PostQuitMessage(0); Exit; end;
        WM_COMMAND: if HWnd(LParam)=hBtnExit then begin PostQuitMessage(0); Exit; end
            else if HWnd(LParam)=hBtnAbout then MessageBox(Window,
                'Вікно, створене функціями API, без використання VCL Delphi.'+#13#13+
                'Copyright© Microsoft... і т.д.', 'API Window',MB_OK or MB_ICONINFORMATION)
            else if HWnd(LParam)=hBtnDlg then begin
                if DialogBox(hInstance,'ICDialog',hWindow,@DlgProc)=IDOK then ChangeParams;
            end;
        WM_KEYDOWN: if (WParam=VK_RETURN) or (WParam=VK_ESCAPE) then
            SendMessage(hBtnExit,BM_CLICK,0,0)
            else if WParam=VK_F1 then SendMessage(hBtnAbout,BM_CLICK,0,0);
    end;
    WindowProc:=DefWindowProc(Window, AMessage, WParam, LParam);
end;

function WinRegister: boolean;
var WindowClass: TWndClass;
begin
    with WindowClass do begin
        Style:=CS_HREDRAW or CS_VREDRAW;
        lpfnWndProc:=@WindowProc;
        cbClsExtra:=0;
        cbWndExtra:=0;
        hInstance:=HInstance;
        hIcon:=LoadIcon(0, IDI_APPLICATION);
        hCursor:=LoadCursor(0, IDC_ARROW);
        hbrBackGround:=COLOR_WINDOW;
        lpszMenuName:=nil;
        lpszClassName:=AppName;
    end;
    Result:=RegisterClass(WindowClass)>0;
end;

function WinCreate: HWND;
var hWindow : HWND;
begin
    hWindow:=CreateWindow(AppName, 'Вікно, створене з використанням API',WS_OVERLAPPEDWINDOW,
        200, 200, 600, 185, 0, 0, HInstance, nil);

    if hWindow<>0 then
        begin

```



```

    ShowWindow(hWindow, SW_SHOWNORMAL);
    UpdateWindow(hWindow);
end;
hBtnExit:=CreateWindow('BUTTON', 'Вихід',WS_CHILD or BS_DEFPUSHBUTTON or WS_TABSTOP, 500, 10,
90, 30, hWindow, 0, HInstance, nil);
if hBtnExit<>0 then ShowWindow(hBtnExit, SW_SHOWNORMAL);
SendMessage(hBtnExit,WM_SETFONT,

CreateFont(18,0,0,0,700,0,0,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUAL
ITY,DEFAULT_PITCH,'Times New Roman Cyr'),1);
hBtnDlg:=CreateWindow('BUTTON', 'Діалог',WS_CHILD or WS_TABSTOP, 500, 75, 90, 30, hWindow, 0,
HInstance, nil);
if hBtnDlg<>0 then ShowWindow(hBtnDlg, SW_SHOWNORMAL);
hBtnAbout:=CreateWindow('BUTTON', 'Про вікно...',WS_CHILD or WS_TABSTOP, 500, 115, 90, 30,
hWindow, 0, HInstance, nil);
if hBtnAbout<>0 then ShowWindow(hBtnAbout, SW_SHOWNORMAL);

hText1:=CreateWindow('STATIC', 'Перший параметр ще не задано',WS_CHILD, 10, 15, 450, 20,
hWindow, 0, HInstance, nil);
if hText1<>0 then ShowWindow(hText1, SW_SHOWNORMAL);
hText2:=CreateWindow('STATIC', 'Другий параметр також ще не задано',WS_CHILD, 10, 40, 450,
20, hWindow, 0, HInstance, nil);
if hText2<>0 then ShowWindow(hText2, SW_SHOWNORMAL);
Result:=hWindow;
end;

begin
if not WinRegister then
begin
    MessageBox(0,                                     // обробник батьківського вікна
                'Клас вікна не зареєстровано',        // адреса тексту повідомлення
                'API Window',                          // адреса заголовку вікна повідомлення
                MB_OK);                                // стиль вікна повідомлення

    Exit;
end;
hWindow:=WinCreate;
if hWindow=0 then
begin
    MessageBox(0, 'Не вийшло створити вікно.', 'API Window', MB_OK);
    Exit;
end;

GetMem(ParamCB,50);

while GetMessage(AMessage,0,0,0) do
begin
    TranslateMessage(AMessage);
    DispatchMessage(AMessage);
end;

FreeMem(ParamCB);

Halt(AMessage.wParam);
end.

```


Лабораторна робота №3

Тема: Системні засоби для роботи з динамічними бібліотеками.

Мета: Вивчення принципів роботи з динамічними бібліотеками за допомогою засобів Windows API на прикладі функцій лінійки задач (Taskbar). Вивчення методів використання динамічних бібліотек у API-базованих програмах. Ознайомлення з принципами використання функцій API, що залежать від версії та платформи ОС.

1. Побудова програми з інтерфейсом у виді піктограми на лінійці задач

Лінійка задач є стандартним елементом провідника (**explorer**) **Windows** та призначена для швидкого доступу до ресурсів комп'ютера (кнопка "Пуск") та запущених програм. Крім цього, програми можуть поміщати додаткову інформацію у системну область лінійки задач у виді невеличких піктограм чи короткого тексту (рис. 3.1). Показ кожної піктограми може супроводжуватись маленькою підказкою, яка з'являється при русі курсора мишки над нею.



Рис. 3.1. Системна область лінійки задач.

Так, програма може поміщати свою піктограму на лінійці задач для індикації своєї роботи, швидкого звертання до її вікна, посилання йому певних повідомлень, виклику контекстного меню програми і т.д. Реалізація доступу до лінійки задач здійснюється за допомогою певних засобів **Windows API**, які і будуть розглянуті у даному документі.

Оскільки на лінійку задач, як правило, поміщають програми, які виконуються у фоновому режимі і не повинні споживати багато ресурсів комп'ютера, то їх вигідно створювати за допомогою засобів **Windows API**. Якщо ж вони мають необхідність звертання до інших, більш громіздких модулів, ці модулі можна оформити у виді динамічних бібліотек ¹⁶ та при необхідності завантажувати у пам'ять і після використання вивільняти.

Нижче розглянуто демонстраційну програму, що не виконує ніякого функціонального навантаження, лише поміщає на лінійку задач свою піктограму, через яку можна викликати контекстне меню і з нього вибрати наступні дії: показати вікно з текстом повідомлення про авторські права, показати вікно для зміни підказки піктограми на лінійці задач, або закінчити роботу. Найбільшу складність реалізації при цьому має процедура зміни підказки для значка, так як для цього слід спроектувати окреме діалогове вікно. Таке вікно було створене стандартними засобами **Delphi** і функцію для його показу скопійовано в окрему динамічну бібліотеку, яка при виклику відповідної команди завантажувється у пам'ять. Сама програма не використовує **VCL Delphi**, і її виконавчий файл має розмір 12,5 Кб ¹⁷. В той же час створена за допомогою **VCL** динамічна бібліотека, яка показує лише одне діалогове вікно і виконує менше корисної дії, має розмір 386 Кб.

1.1. Розробка основного виконавчого модуля програми

Глобальні змінні та константи, які використовуються у програмі, приведено у таблиці №3.1.

Таблиця №3.1

Змінна (константа)	Тип	Призначення
Константи		
Hint	array of char	Містить текст підказки, яка з'являтиметься над піктограмою програми у лінійці задач.
WM_TASKBAR	Ціле число	Ідентифікатор повідомлення, яке буде посилатись від піктограми до головного вікна програми.
ICON_ID	Ціле число	Ідентифікатор піктограми на панелі задач.

¹⁶ DLL (Dynamic-Link Libraries) - динамічно завантажувані бібліотеки.

¹⁷ При компіляції за допомогою **Delphi 6 Enterprise**.

Змінна (константа)	Тип	Призначення
sClassName	String	Назва класу головного вікна.
AboutText	String	Текст про авторські права (буде показуватись у відповідному вікні повідомлення).
AboutCaption	String	Заголовок вікна про авторські права.
Змінні		
hWnd	THandle	Ідентифікатор головного вікна.
WndClass	TWndClass	Структура класу головного вікна.
Msg	TMsg	Повідомлення, яке приймається програмою.
TaskBarCreated	Integer	Ідентифікатор повідомлення про створення лінійки задач.
FormRunning	Boolean	Змінна, яка показує, чи видиме в даний момент діалогове вікно з динамічної бібліотеки.

Розглянемо основний програмний блок проекту ¹⁸:

```
begin
  FillChar(WndClass,SizeOf(WndClass),0);
  with WndClass do
  begin
    // Задаємо параметри класу для головного вікна
    hInstance      := SysInit.hInstance;
    lpzClassName   := sClassName;
    lpfnWndProc    := @WindowProc;
  end;
  // Реєструємо клас вікна
  RegisterClass(WndClass);
  // Створюємо головне вікно програми. Воно мусить бути, але
  // так як воно буде невидимим, то задаємо йому нульовий розмір.
  hWnd:=CreateWindow(sClassName,'',0,0,0,0,0,0,hInstance,NIL);
  if hWnd=0 then // Якщо не вийшло, то виходим з програми
  begin
    MessageBox(0,'Ініціалізацію не проведено!',NIL,ID_OK);
    Exit;
  end;
  // Реєструємо повідомлення про перезапуск Explorer'a
  TaskBarCreated:=RegisterWindowMessage('TaskbarCreated');
  // Створюємо значок
  CreateTaskBarIcon;
  // Ховаємо вікно
  ShowWindow(hWnd, SW_HIDE);
  // Цикл обробки повідомлень
  while GetMessage(Msg,0,0,0) do
  begin
    TranslateMessage(Msg);
    DispatchMessage(Msg);
  end;
  // Забираємо значок при виході
  TaskBarDeleteIcon(hWnd, ICON_ID);
  // Виходим
  Halt(Msg.wParam);
end.
```

Будь-яка програма в середовищі **Windows** повинна мати головне вікно, яке прийматиме повідомлення від операційної системи чи користувача. Так як для реалізації поставленого завдання нам не потрібно показувати вікон, то головним можна зробити невидиме вікно з нульовими розмірами.

Перед використанням структура з інформацією про клас вікна обнуляється шляхом заповнення ділянки пам'яті, яку вона займає, символами з кодом 0. Це здійснює функція **FillChar**. Після цього структура класу вікна заповнюється лише необхідною для роботи інформацією – задається ідентифікатор екземпляра програми, якому належить вікно (**hInstance**), назва класу (**lpzClassName**), та вказівник на віконну функцію (**lpfnWndProc**). Інші властивості для невидимого вікна не потрібні.

¹⁸ Повністю текст програми та супутніх модулів приведено у пункті 3.

Оскільки єдиним елементом інтерфейсу програми є піктограма на лінійці задач, то слід подбати про "надійність" цього каналу зв'язку з користувачем. Так, у роботі системної програми **explorer** (яка відповідає за показ на екрані і функціональність лінійки задач) можуть виникнути різні збої, як з вини запущених процесів, так і в силу внутрішніх особливостей операційної системи **Windows**. При цьому лінійка задач буде перезавантажена, і всі піктограми з неї зникнуть, відізвавши доступ до нашої програми. Тому після реєстрації класу вікна і його створення за допомогою функції **RegisterWindowMessage** реєструємо повідомлення про створення лінійки задач, при надходженні якого слід ще раз розмістити піктограму у системній області.

Для роботи з піктограмами на лінійці задач використовують функцію **API Shell_NotifyIcon**. Набір процедур, призначених для того, щоб додати, модифікувати або забрати піктограму з панелі задач, зібрано в модулі **Taskbar.pas** (див. пункт 3). Функція для створення піктограми має наступний вид:

```
function TaskBarAddIcon(hWindow:THandle; ID:Cardinal; ICON:hIcon; CallbackMessage:Cardinal;
                      Tip:PChar):Boolean;
var NID: TNotifyIconData;
begin
  FillChar(NID,SizeOf(TNotifyIconData),0);
  // Заповнюємо структуру типу TNotifyIconData інформацією про іконку
  with NID do
  begin
    cbSize := SizeOf(TNotifyIconData); // Розмір структури
    Wnd     := hWindow;                // Ідентифікатор головного вікна
    uID     := ID;                     // Ідентифікатор іконки
    uFlags  := NIF_MESSAGE or NIF_ICON or NIF_TIP; // Флажки показу
    uCallbackMessage := CallbackMessage; // Повідомлення від іконки
    hIcon   := Icon;                  // Обробник іконки
    // Із-за того, що szTip має тип масива символів, а Tip- рядок типу PChar,
    // треба скопіювати Tip у szTip за допомогою спеціальної функції lstrcpyn
    lstrcpyn(szTip,Tip,SizeOf(szTip)); // Підказка іконки
  end;
  // Викликаємо стандартну функцію Windows Shell_NotifyIcon для створення
  // іконки на панелі задач
  Result:=Shell_NotifyIcon(NIM_ADD,@NID);
end;
```

У цю функцію передаються такі параметри: **hWindow** – ідентифікатор вікна, яке прийматиме повідомлення від піктограми; **ID** – ідентифікатор піктограми (довільне ціле число, яке визначається програмістом); **ICON** – ідентифікатор піктограми; **CallbackMessage** – ідентифікатор повідомлення, яке посилатиметься від піктограми до вікна, визначеного параметром **hWindow**.

Для модифікації існуючої піктограми використано функцію **TaskBarModifyIcon**:

```
function TaskBarModifyIcon(hWindow:THandle; ID:Cardinal; Flags:Cardinal;
                          ICON:hIcon; Tip:PChar):Boolean;
var NID: TNotifyIconData;
begin
  FillChar(NID, SizeOf(TNotifyIconData), 0);
  with NID do begin
    cbSize := SizeOf(TNotifyIconData);
    Wnd     := hWindow;
    uID     := ID;
    uFlags  := Flags;
    hIcon   := Icon;
    lstrcpyn(szTip, Tip, SizeOf(szTip));
  end;
  // Викликаємо стандартну функцію Windows Shell_NotifyIcon для зміни
  // іконки на Панелі Задач
  Result := Shell_NotifyIcon(NIM_MODIFY, @NID);
end;
```

Параметр **Flags** цієї функції є масивом прапорців, які можуть поєднуватися логічним оператором **OR** та показують, які із наступних значень структури **TNotifyIconData** задано. Елемент може приймати такі значення:

NIF_ICON Означає, що задано елемент структури **hIcon**.
NIF_MESSAGE Означає, що задано елемент структури **uCallbackMessage**.
NIF_TIP Означає, що задано елемент структури **szTip[64]**.

Для знищення піктограми з лінійки задач використано функцію **TaskBarDeleteIcon**:

```
function TaskBarDeleteIcon(hWindow:THandle; ID:Integer):Boolean;
var NID: TNotifyIconData;
begin
  FillChar(NID,SizeOf(TNotifyIconData),0);
  with NID do
  begin
    cbSize := SizeOf(TNotifyIconData);
    Wnd     := hWindow;
    uID     := ID;
  end;
  // Викликаємо стандартну функцію Windows Shell_NotifyIcon для створення
  // іконки на Панелі Задач
  Result := Shell_NotifyIcon(NIM_DELETE,@NID);
end;
```

Ця функція отримує всього два параметри – ідентифікатор вікна **hWindow**, якому належить піктограма, та ідентифікатор самої піктограми **ID**.

Як видно з текстів функцій **TaskBarAddIcon**, **TaskBarModifyIcon**, **TaskBarDeleteIcon**, для здійснення всіх операцій над піктограмами лінійки задач використовується одна і та ж функція – **Shell_NotifyIcon**. Вид виконуваної операції (вставка піктограми, її модифікація чи знищення) задається її першим параметром.

Для того, щоб створити контекстне меню, яке з'являтиметься над піктограмою, використаємо мову сценаріїв ресурсів (**Resource Script Language**). Вид сценарію для простого меню з трьох елементів має такий вид:

```
#include "constant.pas"
MAINMENU MENU
BEGIN
  POPUP "Dummy" BEGIN
    MENUITEM "&Про програму", ID_ABOUT
    MENUITEM "&Форма з DLL", ID_DLLFORM
    MENUITEM SEPARATOR
    MENUITEM "&Вихід", ID_CLOSE
  END
END
```

Призначення використаних елементів синтаксису приведено у таблиці №3.2.

Таблиця №3.2

Директива, команда	Синтаксис	Призначення
#include	#include "filename" або #include <filename>	При компіляції на місці директиви вставляється файл "filename". Якщо "filename" є модулем Pascal, то він повинен містити лише список констант. Приклад: #include "constant.pas"
MENU	resource-name MENU BEGIN item-definitions END	Багаторядкова конструкція, яка оголошує ресурс меню з назвою resource-name разом з елементами меню, які оголошуються всередині блоку BEGIN... END .

Директива, команда	Синтаксис	Призначення
POPUP	POPUP [popup-name] [popup-attributes] BEGIN item-definitions END	Оголошує ресурс контекстного меню з назвою popup-name, може мати атрибути popup-attributes.
MENUITEM	MENUITEM [item-text] [item-ID] [item-attributes] або MENUITEM SEPARATOR	Оголошує для ресурсу меню елемент меню з текстом item-text, ідентифікатором item-ID, може також містити атрибути item-attributes. Якщо у меню повинен бути вставлений розподільювач, використовується параметр SEPARATOR. Приклад: MENUITEM "&Вихід", ID_CLOSE

У даний сценарій вставлено файл **Constant.pas**¹⁹, який містить список ідентифікаторів меню і використовується як сценарієм ресурсів, так і головною програмою. Цим досягається узгодженість ідентифікаторів елементів меню для сценарію та проекту. Скомпільований файл ресурсів слід під'єднати до проекту **Delphi**. Якщо його назва **TrayRes.res**, то це здійснюється так:

```
{$R TrayRes.res}
```

Для показу меню використовується процедура **PopupMenu**:

```
procedure PopupMenu(hWnd: THandle);
var
  Menu : hMenu;    // Ідентифікатор меню з ресурса
  Popup : hMenu;    // Ідентифікатор елемента меню
  P : TPoint;
begin
  Menu:=LoadMenu(hInstance, 'MAINMENU');
  Popup:=GetSubMenu(Menu, 0);
  GetCursorPos(P);
  SetForegroundWindow(hWnd);
  // Показуємо контекстне меню на іконці у Панелі Задач
  TrackPopupMenu(Popup, TPM_CENTERALIGN or TPM_LEFTBUTTON, P.X, P.Y, 0, hWnd, NIL);
  // Ховаємо меню, якщо з нього не було вибрано жодної команди,
  // і якщо вікно меню перестало бути активним
  PostMessage(hWnd, WM_NULL, 0, 0);
  // Знищуємо меню
  DestroyMenu(Menu);
end;
```

Спочатку меню завантажується з ресурсу за допомогою функції **LoadMenu** (при цьому його ідентифікатор зберігається у змінній **Menu**), далі за функцією **GetSubMenu** отримується ідентифікатор першого елемента меню та обчислюється позиція курсора мишки (функція **GetCursorPos**, результат зберігається у змінній **P** типу **TPoint**). Далі активізується головне вікно програми – воно "ставиться" на передній план понад всіма іншими вікнами (**SetForegroundWindow**)²⁰. Це здійснюється для того, щоб меню (яке належить головному вікну) не могло бути перекрите іншими вікнами у випадку неактивності нашої програми. Відображується меню на екрані в заданій точці **P** (тобто там, де був курсор миші при його виклику) за допомогою функції **TrackPopupMenu**. Після закриття воно знищується, і вивільняється вся пов'язана з ним область пам'яті.

Повідомлення від піктограми на лінійці задач надсилаються до віконної функції головного вікна. Вона має такий вигляд:

```
function WindowProc(hWnd: THandle; uMsg, wParam, lParam: Integer): Integer; stdcall; export;
type TDestroyDLLForm = procedure;
     TForegroundDLLForm = procedure;
var DestroyDLLForm : TDestroyDLLForm;
```

¹⁹ Див. пункт 3.

²⁰ Хоча слід зауважити, що воно невидиме, тому його все одно не буде показано.

```

    ForegroundDLLForm : TForegroundDLLForm;
    hDLL : THandle;
begin
    // Якщо отримано зареєстроване нами повідомлення про перезапуск провідника
    // Windows, то створюємо іконку до своєї програми ще раз
    if uMsg=TaskBarCreated then CreateTaskbarIcon;
    case uMsg of
        // Повідомлення від меню
        WM_COMMAND:
            case wParam of
                // Повідомлення від команди меню "Вихід"
                ID_CLOSE : PostMessage(hWnd,WM_DESTROY,0,0); // Посилаємо головному
                                                                // вікну повідомлення
                                                                // про вихід

                // Повідомлення від команди меню "Про програму"
                ID_ABOUT : ShowAboutDialog; // Показати діалог "Про програму"
                // Повідомлення від команди меню "Форма з DLL"
                ID_DLLFORM: ShowDLLForm(hWnd); // Показати вікно з DLL
            end;
        // Повідомлення від іконки на панелі задач
        WM_TASKBAR:
            case wParam of
                ICON_ID:
                    case lParam of
                        // При клацанні лівою кнопкою миші на іконці показуємо вікно з DLL
                        WM_LBUTTONDOWN : if not FormRunning then ShowDLLForm(hWnd) else TopDLLForm;
                        // При клацанні правою кнопкою миші показуємо контекстне меню
                        WM_RBUTTONDOWN : PopupMenu(hWnd);
                    end;
            end;
        // Повідомлення про вихід
        WM_DESTROY:
            begin
                // Якщо діалогове вікно з DLL показується, знищуємо його
                if FormRunning then DestructDLLForm;
                PostQuitMessage(0); // Посилаємо головному вікну повідомлення про вихід
            end;
        end;
        // Інші повідомлення Windows
        Result:=DefWindowProc(hWnd,uMsg,wParam,lParam);
    end;
end;

```

Дана функція обробляє наступні повідомлення:

TaskBarCreated	Отримується при створенні лінійки задач і використовується для малювання піктограми на новоствореній панелі.
WM_COMMAND	Отримується при виборі елемента контекстного меню. Меню має три команди, і залежно від того, яка з них вибрана, параметр wParam повідомлення прийматиме значення ідентифікатора відповідного елемента меню (ID_CLOSE , ID_ABOUT або ID_DLLFORM ²¹).
WM_TASKBAR	Повідомлення від іконки на панелі задач (див. табл. №3.1). При цьому параметр wParam містить ідентифікатор піктограми (так як з однієї програми можна створити кілька піктограм), а параметр lParam вказує на тип події, яка трапилась з піктограмою, а саме: WM_LBUTTONDOWN - натиснено ліву кнопку миші, WM_RBUTTONDOWN - натиснено праву кнопку миші, WM_MOUSEMOVE - миш рухається над піктограмою.
WM_DESTROY	Отримується при знищенні вікна. Використовується для того, щоб у випадку видимості діалогового вікна з динамічної бібліотеки (описане пізніше) закрити при виході і його.

²¹ Див. текст файлу Constant.pas.

Якщо функція отримує від меню повідомлення типу **WM_COMMAND** з параметром **ID_ABOUT**, за допомогою процедури **ShowAboutDialog** здійснюється вивід на екран простого повідомлення про авторські права на програму.

```

procedure ShowAboutDialog;
var
  Version      : TOSVersionInfo; // Змінна для отримання версії ОС
  MsgBoxParamsW : TMsgBoxParamsW; // Параметри для вікна повідомлення під WinNT
  MsgBoxParamsA : TMsgBoxParamsA; // Параметри для вікна повідомлення під Win9x
begin
  // Отримуємо інформацію про версію ОС
  Version.dwOSVersionInfoSize:=SizeOf(TOSVersionInfo);
  GetVersionEx(Version);
  if Version.dwPlatformId=VER_PLATFORM_WIN32_NT then
  begin // Якщо Windows NT, 2000, XP
    // Обнуляємо структуру (record) з параметрами повідомлення
    FillChar(MsgBoxParamsW,SizeOf(MsgBoxParamsW),#0);
    // Задаємо параметри повідомлення
    with MsgBoxParamsW do
    begin
      cbSize:=SizeOf(MsgBoxParamsW);
      hwndOwner:=hWnd;
      hInstance:=SysInit.hInstance;
      lpszText:=AboutText;
      lpszCaption:=AboutCaption;
      lpszIcon:='MAINICON';
      dwStyle:=MB_USERICON;
    end;
    // Показуємо повідомлення під WinNT
    MessageBoxIndirectW(MsgBoxParamsW);
  end
  else begin // Якщо Windows 95, 98, ME...
    // Обнуляємо структуру (record) з параметрами повідомлення
    FillChar(MsgBoxParamsA,SizeOf(MsgBoxParamsA),#0);
    // Задаємо параметри повідомлення
    with MsgBoxParamsA do
    begin
      cbSize:=SizeOf(MsgBoxParamsA);
      hwndOwner:=hWnd;
      hInstance:=SysInit.hInstance;
      lpszText:=AboutText;
      lpszCaption:=AboutCaption;
      lpszIcon:='MAINICON';
      dwStyle:=MB_USERICON;
    end;
    // Показуємо повідомлення під Win9x
    MessageBoxIndirectA(MsgBoxParamsA);
  end;
end;

```

Оскільки для виводу тексту повідомлення використовується функція **MessageBoxIndirect**, яка по-різному працює для різних платформ **Windows**, то спочатку за допомогою функції **GetVersionEx** здійснюється визначення версії операційної системи. Для цього у функцію **GetVersionEx** передається структура **Version** типу **TOSVersionInfo**, у поле **dwPlatformId** якої і поміщається необхідна інформація. Так, якщо програма запущена під операційною системою **Windows NT, 2000** або **XP**, вказане поле рівне **VER_PLATFORM_WIN32_NT**. На системах типу **Windows 95, 98, ME** воно прийме значення **VER_PLATFORM_WIN32_WINDOWS**. В залежності від платформи ОС далі використовуються відповідні модифікації функції **MessageBoxIndirect**:

MessageBoxIndirectW – для **Windows NT, 2000** або **XP**;

MessageBoxIndirectA – для **Windows 95, 98, ME**.

При цьому для передачі параметрів вікна повідомлення заповнюються відповідно структури **MsgBoxParamsW** або **MsgBoxParamsA**.

Якщо віконна функція головного вікна програми (**WindowProc**) отримує визначене у програмі повідомлення **WM_TASKBAR** від піктограми на лінійці задач, то здійснюється перевірка значення

параметра **wParam** повідомлення, який вказує на тип події, що трапилась з піктограмою. Так, при натисканні правою кнопкою миші на іконці (**wParam=WM_RBUTTONDOWN**) за допомогою вже розглянутої процедури **PopupMenu** здійснюється розкривання контекстного меню. При натисканні лівою кнопкою миші на піктограмі (**wParam=WM_LBUTTONDOWN**) здійснюється показ діалогового вікна з динамічної бібліотеки, яке містить одне текстове поле для зміни рядка підказки піктограми на лінійці задач (див. рис. 3.2). Сама динамічна бібліотека описана у розділі 1.2.

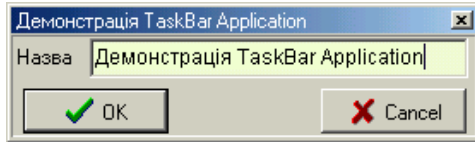


Рис. 3.2. Діалогове вікно з динамічної бібліотеки.

При цьому, якщо діалогове вікно ще не було створене (глобальна змінна **FormRunning=false**), воно створюється за допомогою процедури **ShowDLLForm**. Якщо ж воно вже показане (**FormRunning=true**), процедурою **TopDLLForm** ставимо його на передній план (так як воно може перекритись іншими вікнами).

Останнє повідомлення, яке обробляється віконною функцією **WindowProc** – це повідомлення про знищення головного вікна **WM_DESTROY**. Якщо діалогове вікно з динамічної бібліотеки відображається на екрані, воно повинно бути закрито, що і реалізується процедурою **DestructDLLForm**.

Процедури **ShowDLLForm**, **TopDLLForm** та **DestructDLLForm** містять виклики до динамічної бібліотеки з діалоговим вікном **UI.dll** і будуть описані у розділі 1.3.

1.2. Розробка динамічної бібліотеки з діалоговим вікном

Текст проекту динамічної бібліотеки та супутніх модулів в повному обсязі приведено у пункті 3. З бібліотеки експортуються три об'єкти: функція **DoDLLForm** (для показу діалогового вікна), процедура **ForegroundDLLForm** (для розміщення діалогового вікна на передній план) та процедура **DestroyDLLForm** (для знищення діалогового вікна).

Функція просто створює діалогове вікно стандартним для **Delphi** способом та повертає значення **true**, якщо у ньому було натиснено кнопку "OK". Її текст наступний:

```
function DoDLLForm (lpNewHint:PChar; iSize:Integer):Boolean;
begin
  fUI:=TfUI.Create(nil);
  with fUI do
  try
    Edit1.Text:=lpNewHint;
    Edit1.MaxLength:=iSize-1;
    Caption:=lpNewHint;
    SetForegroundWindow(fUI.Handle);
    Result:=ShowModal=mrOk;
    if Result then StrPCopy(lpNewHint, Edit1.Text);
  finally
    if fUI<>nil then Free;
  end;
end;
```

У випадку натискання кнопки "OK" (**Result=true**) текстовий рядок типу **string**, введений у поле редагування **Edit1** (типу **TEdit**), копіюється у змінну, задану вказівником **lpNewHint** (який передається як перший параметр функції і є вказівником на рядок з кінцевим нулем). Перетворення стандартного для **Pascal** типу **string** у стандартний для **Windows API** рядок **PChar** здійснюється функцією **StrPCopy**.

Процедура для переміщення вікна на передній план має такий вид:

```
procedure ForegroundDLLForm;
begin
  if Assigned(fUI) then SetForegroundWindow(fUI.Handle);
end;
```


Знищення діалогового вікна здійснюється процедурою **DestroyDLLForm**:

```
procedure DestroyDLLForm;
begin
  if Assigned(fUI) then fUI.Close;
end;
```

Процедури **ForegroundDLLForm** та **DestroyDLLForm** спочатку за допомогою функції **Assigned** перевіряють, чи існує діалогове вікно взагалі, оскільки спроба звертання за нульовим вказівником (а **fUI** – це вказівник на об'єкт класу **TfUI**) приведе до помилки **Windows** типу "Access violation..." ("Ошибка обращения к памяти...").

1.3. Звертання до функцій DLL з основного модуля

Основний модуль програми (описаний у розділі 1.1) містить три процедури, які використовують звертання до динамічної бібліотеки з діалоговим вікном: **ShowDLLForm**, **TopDLLForm** та **DestructDLLForm**.

Розглянемо принцип використання **DLL** на прикладі процедури **ShowDLLForm** (яка показує діалогове вікно шляхом виклику функції бібліотеки **DoDLLForm**):

```
procedure ShowDLLForm(hWnd: THandle);
type
  // Прототип функції з DLL. Напрямку написати Uses не варто, так як
  // при цьому в проект скомпілюється і форма, що збільшить його розмір.
  TDoDLLForm=function(lpNewHint:PChar; iSize:Integer):Boolean; stdcall;
var
  hDLL      : THandle;           // Handler (ідентифікатор) динам. бібліотеки
  DoDLLForm : TDoDLLForm;       // Функція для створення вікна з DLL
begin
  // Якщо вікно з DLL вже показано, то виходимо
  if FormRunning then Exit;
  // Завантажуємо DLL
  hDLL:=LoadLibrary('UI.DLL');
  // Якщо не вийшло, пишем про це і припиняємо спробу показу вікна
  if hDLL=0 then MessageBox(0, 'Бібліотеку UI.DLL не знайдено', NIL, MB_OK)
  else begin
    // Якщо DLL завантажено, отримуємо адресу функції DoDLLForm
    // (див. проект бібліотеки UI.dpr), яка показує вікно з DLL
    DoDLLForm:=GetProcAddress(hDLL, 'DoDLLForm');
    if Assigned(DoDLLForm) then // Якщо таку функцію знайдено, то ...
    begin
      FormRunning:=TRUE;
      try
        // Пробуємо показати вікно форми з DLL
        if DoDLLForm(@Hint, SizeOf(Hint)) then
        begin
          // Модифікуємо іконку на Панелі Задач (а саме- її підказку),
          // якщо користувач натиснув кнопку "OK" у формі з DLL.
          TaskBarModifyIcon(hWnd, ICON_ID, NIF_TIP, 0, Hint);
        end;
      finally
        // В будь-якому випадку змінний FormRunning присвоюємо значення FALSE
        // що свідчить про те, що вікно з DLL не показується.
        FormRunning:=FALSE;
      end;
    end;
  end;
  // Вивантажуємо DLL з пам'яті
  FreeLibrary(hDLL);
end;
```

Перед викликом функції (процедури) з **DLL** слід оголосити відповідний процедурний тип даних. Так, функція **DoDLLForm**, розміщена в бібліотеці, має наступне оголошення:

```
function DoDLLForm (lpNewHint:PChar; iSize:Integer):Boolean; stdcall; export;
```

Тому перед її викликом оголошуємо наступний тип даних:

```
type TDoDLLForm=function(lpNewHint:PChar; iSize:Integer):Boolean; stdcall;
```

При цьому кількість та тип параметрів оголошеного процедурного типу повинні співпадати з кількістю та типом даних функції з **DLL**. Далі створюємо змінну даного типу, яка призначена для отримання адреси точки входу до функції з **DLL**:

```
var  
  DoDLLForm22 : TDoDLLForm;
```

Після цього слід завантажити у пам'ять бібліотеку, яка містить бажану функцію. Ця дія виконується функцією **LoadLibrary**, що розміщує заданий виконавчий модуль в адресному просторі викликаючого процесу і повертає його ідентифікатор. Якщо виявляється, що модуль вже завантажено, то функція збільшує на одиницю системний лічильник посилань на нього. Після цього за допомогою функції **GetProcAddress** слід отримати адресу точки входу в бажану функцію (процедуру) і запам'ятати її у відповідній змінній:

```
DoDLLForm:=GetProcAddress(hDLL, 'DoDLLForm');
```

Якщо операція обчислення адреси пройшла успішно, то відповідну функцію можна викликати таким чином:

```
... DoDLLForm(@Hint,SizeOf(Hint)) ...
```

Проте, у випадку невдалого звертання до функції **GetProcAddress** адресу бажаної функції з ряду причин може не бути отримано. Тоді виконання приведеного вище коду приведе до загальної помилки доступу до пам'яті. Тому перед звертанням до функції за заданою адресою слід перевірити, чи дійсно відповідній змінній (**DoDLLForm**) було присвоєно адресу:

```
if Assigned(DoDLLForm) then ...
```

Після виконання відповідних операцій з функцією **DLL**, саму бібліотеку можна вивантажити з пам'яті, звільнивши певні ресурси комп'ютера. Це здійснює функція **FreeLibrary**:

```
FreeLibrary(hDLL);
```

Ця функція зменшує на одиницю лічильник посилань на вказану бібліотеку, і якщо він стає рівен нулю, вивільняє з пам'яті саму **DLL**.

Аналогічним чином реалізовано і звертання до двох інших процедур у динамічній бібліотеці-**TopDLLForm** та **DestructDLLForm**.

2. Завдання на лабораторну роботу

Написати (з використанням лише функцій **Windows API**, без впровадження компонентів **VCL Delphi**) програму, яка матиме інтерфейс у виді піктограми на панелі задач, здійснюватиме розрахунок арифметичного виразу згідно варіанту (таблиця №3.3) та відображатиме його в діалоговому вікні, яке викликатиметься з динамічної бібліотеки (**DLL** може бути спроектована за допомогою **VCL Delphi**). Значення змінних, що використовуються для розрахунку, також повинні задаватися за допомогою діалогового вікна з динамічної бібліотеки. Програма повинна

²² Зазначимо, що назва даної змінної процедурного типу може не співпадати з назвою функції у динамічній бібліотеці.

здійснювати перевірку можливості обчислення виразу при заданих параметрах та у випадку неможливості проведення обчислень видавати відповідне повідомлення²³.

Таблиця №3.3

Варіант	Завдання
1	$y = 78a^6 + 32a^4 - 23ab^2$
2	$y = \frac{- a^3 + 4b^5 }{\sqrt[3]{3a^2b}}$
3	$y = \frac{\sqrt[4]{a+3b}}{(a-b)(3b+2a)}$
4	$y = \frac{\ln(3\pi a^2)}{\cos(2ba + 3b^{1/3})}$
5	$y = \frac{\sqrt[5]{a + (3b^3 - a^2)}}{4.36b - 0.23a}$
6	$y = \sqrt[5]{(3a - 6b)} + \frac{2a\sqrt{ab}}{6b}$
7	$y = \frac{\lg(3.43a^{1/3} + 4.9b^2)}{\sqrt{3ba}}$
8	$y = 5a^2 - 45\frac{b^2}{-4a^{1/3}} + 3a^3b$
9	$y = \frac{\sqrt{a+b^3}}{\operatorname{tg}(ab^2)}$
10	$y = \cos^2(a) + 3\sin^2(ab)$
11	$y = \frac{\ln(a^3)}{a + 4.23ab + a^2b(1 - \sqrt{b})}$
12	$y = (356a^2 - 23b)(\sqrt{3ab - b})$
13	$y = 2.36\pi^3 + 2.58\cos^5(3ab + 2)$
14	$y = 12a + \sqrt[3]{3ab + 4.25b^3} + \sqrt{0.23a - b}$
15	$y = 2a^5 + \frac{3b^3}{6a^2 - b}$
16	$y = \cos(a + b) \cdot \sqrt{a^3b^5}$
17	$y = \frac{3a^3}{b^2} + \frac{23a + b}{0.6a^2} + \frac{\sqrt{ab}}{a^3}$
18	$y = 2a^3 + 3b^5 - \frac{6a + b}{3a^2}$
19	$y = \frac{-2ab - \sqrt{a + b}}{3a(4a^3 + 3b^2)}$

²³ Наприклад, квадратний корінь з від'ємного числа не може бути обчислено.

Варіант	Завдання
20	$y = -4a + 6b^3 - 34a^3b^4 - \frac{a}{b}$
21	$y = \sqrt{\frac{x_2^2 + x_1 / x_2}{16x_2x_1}}$
22	$y = 23\cos^2(x_1^3x_2^5) + 2x_1$
23	$y = \sin^2\left(x_1 \frac{x_2}{x_1 + 53x_2^2}\right)$
24	$y = 45x_1 \sin x_2 + \sqrt{9x_2x_1^3}$
25	$y = 0.1x_1 \sin x_2 \cos x_1^4 + 55$
26	$y = \operatorname{tg}(x_1 - x_2^2) + 31.55x_2x_1^2$
27	$y = \sin(x_1 - x_2^3 + \sqrt{x_1}) - 1.3x_1^3$
28	$y = \cos(\sqrt{x_2} + 34x_1) - 4\sin(x_2)$
29	$y = \frac{4\sin(3 + x_1x_2)}{34 - 9x_2^3}$
30	$y = \frac{\cos^2\left(\lg_{10} \frac{x_2}{x_1}\right)}{45 + x_2}$

3. Програмні тексти

Текст файлу проекту програми (TrayIcon.dpr).

```
( *****
Цей приклад демонструє створення програми з інтерфейсом
у вигляді значка в System Tray. Програма написана без
використання VCL і в скомпільованому вигляді займає
12 Кб (у Delphi 6 Enterprise). Призначений для користувача
інтерфейс скомпільований в окрему DLL і підвантажується при потребі.
***** )

program TrayIcon;

uses Windows, Messages, Constant, TaskBar;

{$R *.RES}
{$R TrayRes.RES}

const
  Hint : array[0..63] of Char='Демонстрація TaskBar Application';
  WM_TASKBAR = WM_APP+1;           // повідомлення від Tray Icon
  ICON_ID=0;                       // ідентифікатор значка
  sClassName='sTaskBarHandlerWindow'; // Ім'я класу вікна
  AboutText='Simple TaskBar Application Demo'#13#13 +
    'Copyright© ICSofT, ТДТУ, Кафедра АВ. 2002.';
  AboutCaption='Демонстрація простої TaskBar-програми';

var
  hWnd      : THandle;
  WndClass  : TWndClass;
  Msg       : TMsg;
```

```

TaskBarCreated : Integer;

// Прапорець "форма завантажена". Для запобігання повторному
// завантаженню форми
FormRunning : Boolean=false;

procedure ShowAboutDialog;
var
  Version      : TOSVersionInfo; // Змінна для отримання версії ОС
  MsgBoxParamsW : TMsgBoxParamsW; // Параметри для вікна повідомлення під WinNT
  MsgBoxParamsA : TMsgBoxParamsA; // Параметри для вікна повідомлення під Win9x
begin
  // Функція MessageBoxIndirect, яка використовується для
  // виведення About, по різному працює під Windows 9x і NT.
  // Тому визначаємо спочатку версію Windows.
  // Отримуємо інформацію про версію ОС
  Version.dwOSVersionInfoSize:=SizeOf(TOSVersionInfo);
  GetVersionEx(Version);
  if Version.dwPlatformId=VER_PLATFORM_WIN32_NT then
  begin // Якщо Windows NT, 2000, XP
    // Обнуляємо структуру (record) з параметрами повідомлення
    FillChar(MsgBoxParamsW,SizeOf(MsgBoxParamsW),#0);
    // Задаємо параметри повідомлення
    with MsgBoxParamsW do
    begin
      cbSize:=SizeOf(MsgBoxParamsW);
      hwndOwner:=hWnd;
      hInstance:=SysInit.hInstance;
      lpszText:=AboutText;
      lpszCaption:=AboutCaption;
      lpszIcon:='MAINICON';
      dwStyle:=MB_USERICON;
    end;
    // Показуємо повідомлення під WinNT
    MessageBoxIndirectW(MsgBoxParamsW);
  end
  else begin // Якщо Windows 95, 98, ME...
    // Обнуляємо структуру (record) з параметрами повідомлення
    FillChar(MsgBoxParamsA,SizeOf(MsgBoxParamsA),#0);
    // Задаємо параметри повідомлення
    with MsgBoxParamsA do
    begin
      cbSize:=SizeOf(MsgBoxParamsA);
      hwndOwner:=hWnd;
      hInstance:=SysInit.hInstance;
      lpszText:=AboutText;
      lpszCaption:=AboutCaption;
      lpszIcon:='MAINICON';
      dwStyle:=MB_USERICON;
    end;
    // Показуємо повідомлення під Win9x
    MessageBoxIndirectA(MsgBoxParamsA);
  end;
end;

// Показ форми з динамічної бібліотеки
procedure ShowDLLForm(hWnd: THandle);
type
  // Прототип функції з DLL. Напряму написати Uses не варто, так як
  // при цьому в проект скопілюється і форма, що збільшить його розмір.
  TDoDLLForm=function(lpNewHint:PChar; iSize:Integer):Boolean; stdcall;
var
  hDLL      : THandle; // Handler (ідентифікатор) динам. бібліотеки
  DoDLLForm : TDoDLLForm; // Функція для створення вікна з DLL
begin
  // Якщо вікно з DLL вже показано, то виходимо
  if FormRunning then Exit;
  // Завантажуємо DLL
  hDLL:=LoadLibrary('UI.DLL');
  // Якщо не вийшло, пишемо про це і припиняємо спробу показу вікна
  if hDLL=0 then MessageBox(0, 'Бібліотеку UI.DLL не знайдено', NIL, MB_OK)

```

```

else begin
    // Якщо DLL завантажено, отримуємо адресу функції DoDLLForm
    // (див. проект бібліотеки UI.dpr), яка показує вікно з DLL
    DoDLLForm:=GetProcAddress(hDLL,'DoDLLForm');
    if Assigned(DoDLLForm) then // Якщо таку функцію знайдено, то ...
    begin
        FormRunning:=TRUE;
        try
            // Пробуємо показати вікно форми з DLL
            if DoDLLForm(@Hint,SizeOf(Hint)) then
            begin
                // Модифікуємо іконку на Панелі Задач (а саме- її підказку),
                // якщо користувач натиснув кнопку "OK" у формі з DLL.
                TaskBarModifyIcon(hWnd,ICON_ID,NIF_TIP,0,Hint);
            end;
        finally
            // В будь-якому випадку змінний FormRunning присвоюємо значення FALSE
            // що свідчить про те, що вікно з DLL не показується.
            FormRunning:=FALSE;
        end;
    end;
end;
// Вивантажуємо DLL з пам'яті
FreeLibrary(hDLL);
end;

// Функція поміщає вікно форми з DLL на передній план
procedure TopDLLForm;
type TForegroundDLLForm=procedure;
var hDLL      : THandle;                // Handler (ідентифікатор) динам. бібліотеки
    ForegroundDLLForm : TForegroundDLLForm;
begin
    hDLL:=LoadLibrary('UI.DLL');
    if hDLL<>0 then begin
        ForegroundDLLForm:=GetProcAddress(hDLL,'ForegroundDLLForm');
        if Assigned(ForegroundDLLForm) then ForegroundDLLForm;
    end;
end;

// Процедура для знищення вікна з DLL
procedure DestructDLLForm;
type TDestroyDLLForm=procedure;
var DestroyDLLForm : TDestroyDLLForm;
    hDLL : THandle;
begin
    hDLL:=LoadLibrary('UI.DLL');
    if hDLL<>0 then begin
        DestroyDLLForm:=GetProcAddress(hDLL,'DestroyDLLForm');
        if Assigned(DestroyDLLForm) then
        begin
            DestroyDLLForm;
            FormRunning:=FALSE;
        end;
    end;
end;

// Процедура для створення іконки на Панелі Задач
procedure CreateTaskBarIcon;
begin
    TaskBarAddIcon(hWnd,ICON_ID,LoadIcon(hInstance,'MAINICON'),WM_TASKBAR,Hint);
end;

// Процедура для показу контекстного меню для іконки на панелі задач
procedure PopupMenu(hWnd: THandle);
var
    Menu   : hMenu;    // Обробник меню з ресурса
    Popup  : hMenu;    // Обробник контекстного меню
    P      : TPoint;
begin
    // Меню загружається з ресурса (див. файл TrayRes.RC)

```

```

Menu:=LoadMenu(hInstance,'MAINMENU');
// Отримуємо обробник першого елемента меню (з номером 0)
Popup:=GetSubMenu(Menu,0);
// Отримуємо позицію курсора
GetCursorPos(P);
// Задаємо активне вікно (ним є невидиме головне вікно програми)
// Задається того, що наше контекстне меню відноситься до нього
SetForegroundWindow(hWnd);
// Показуємо контекстне меню на іконці у Панелі Задач
TrackPopupMenu(Popup,TPM_CENTERALIGN or TPM_LEFTBUTTON,P.X,P.Y,0,hWnd,NIL);
// Ховаємо меню, якщо з нього не було вибрано жодної команди,
// і якщо вікно меню перестало бути активним
PostMessage(hWnd, WM_NULL, 0, 0);
// Знищуємо меню
DestroyMenu(Menu);
end;

// Віконна процедура
function WindowProc(hWnd: THandle; uMsg, wParam, lParam: Integer): Integer; stdcall; export;
begin
// Якщо отримано зареєстроване нами повідомлення про перезапуск Провідника
// Windows (таке часом буває при збоях у Windows), то створюємо іконку
// до своєї програми ще раз
if uMsg=TaskBarCreated then CreateTaskbarIcon;
case uMsg of
// Повідомлення від меню
WM_COMMAND:
case wParam of
// Повідомлення від команди меню "Вихід"
ID_CLOSE : PostMessage(hWnd,WM_DESTROY,0,0); // Посилаємо головному
// вікну повідомлення
// про знищення

// Повідомлення від команди меню "Про програму"
ID_ABOUT : ShowAboutDialog; // Показати Діалог "Про програму"
// Повідомлення від команди меню "Форма з DLL"
ID_DLLFORM: ShowDLLForm(hWnd); // Показати вікно з DLL
end;
// Повідомлення від іконки на Панелі Задач
WM_TASKBAR:
case wParam of
ICON_ID:
case lParam of
// При клацанні лівою кнопкою миші на іконці показуємо вікно з DLL
WM_LBUTTONDOWN : if not FormRunning then ShowDLLForm(hWnd) else TopDLLForm;
// При клацанні правою кнопкою миші показуємо контекстне меню
WM_RBUTTONDOWN : PopupMenu(hWnd);
end;
end;
// Повідомлення про вихід
WM_DESTROY:
begin
if FormRunning then DestructDLLForm;
PostQuitMessage(0); // Посилаємо головному вікну повідомлення про вихід
end;
end;
// Інші повідомлення Windows
Result:=DefWindowProc(hWnd,uMsg,wParam,lParam);
end;

begin
FillChar(WndClass,SizeOf(WndClass),0);
with WndClass do
begin
// Задаємо параметри класу для головного вікна
hInstance := SysInit.hInstance;
lpzClassName := sClassName;
lpfnWndProc := @WindowProc;
// Всі інші властивості для невидимого вікна не обов'язкові
end;
// Реєструємо клас вікна
RegisterClass(WndClass);

```

```

// Створюємо головне вікно програми. Воно мусить бути, але
// так як воно непотрібне і буде невидимим, то задаємо йому
// нульовий розмір.
hWnd:=CreateWindow(sClassName,'',0,0,0,0,0,0,hInstance,NIL);
if hWnd=0 then // Якщо не вийшло, то виходим з програми
begin
  MessageBox(0,'Ініціалізацію не проведено!',NIL,ID_OK);
  Exit;
end;
// Реєструємо повідомлення про перезапуск Explorer'a
TaskBarCreated:=RegisterWindowMessage('TaskbarCreated');
// Створюємо значок
CreateTaskBarIcon;
// Ховаємо вікно
ShowWindow(hWnd, SW_HIDE);
// Цикл обробки повідомлень
while GetMessage(Msg,0,0,0) do
begin
  TranslateMessage(Msg);
  DispatchMessage(Msg);
end;
// Забираємо значок при виході
TaskBarDeleteIcon(hWnd, ICON_ID);
// Виходим
Halt(Msg.wParam);
// Кінець. Вийшли.
end.

```

Текст модуля для роботи з піктограмою на панелі задач (TaskBar.pas).

```

unit TaskBar;

interface

uses Windows, ShellAPI;
const
  // Визначені у Windows флажки для іконки
  NIF_TIP = $00000004;
  NIF_ICON = $00000002;

// Функція для створення іконки на Панелі Задач
function TaskBarAddIcon(
  hWnd: THandle;           // ідентифікатор вікна, що створює іконку (значок)
  ID: Cardinal;            // ідентифікатор значка
  ICON: hIcon;             // іконка
  CallbackMessage: Cardinal; // повідомлення, яке буде посилатися вікну від іконки
  Tip: PChar              // Підказка
): Boolean;

// Функція для модифікації іконки на Панелі Задач
function TaskBarModifyIcon(hWnd: THandle; ID: Cardinal; Flags: Cardinal;
  ICON: hIcon; Tip: PChar): Boolean;

// Функція для знищення іконки на Панелі Задач
function TaskBarDeleteIcon(hWnd: THandle; ID: Integer): Boolean;

implementation

// Функція для створення іконки на Панелі Задач
function TaskBarAddIcon(hWnd:THandle; ID:Cardinal; ICON:hIcon; CallbackMessage:Cardinal;
  Tip:PChar):Boolean;
var NID: TNotifyIconData;
begin
  FillChar(NID,SizeOf(TNotifyIconData),0);
  // Заповнюємо структуру типу TNotifyIconData інформацією про іконку
  with NID do
  begin
    cbSize := SizeOf(TNotifyIconData); // Розмір структури
    Wnd := hWnd;                       // Обробник головного вікна
    uID := ID;                         // Ідентифікатор іконки
    uFlags := NIF_MESSAGE or NIF_ICON or NIF_TIP; // флажки показу

```



```

uCallbackMessage := CallbackMessage;           // Повідомлення від іконки
hIcon := Icon;                                 // Обробник іконки
// Із-за того, що szTip має тип масива символів, а Tip- рядок типу PChar,
// треба скопіювати Tip у szTip за допомогою спеціальної функції lstrcpyn
lstrcpyn(szTip,Tip,SizeOf(szTip));             // Підказка іконки
end;
// Викликаємо стандартну функцію Windows Shell_NotifyIcon для створення
// іконки на Панелі Задач
Result:=Shell_NotifyIcon(NIM_ADD,@NID);
end;

// Функція для модифікації іконки на Панелі Задач
function TaskBarModifyIcon(hWindow:THandle; ID:Cardinal; Flags:Cardinal;
                          ICON:hIcon; Tip:PChar):Boolean;
var NID: TNotifyIconData;
begin
  FillChar(NID, SizeOf(TNotifyIconData), 0);
  with NID do begin
    cbSize := SizeOf(TNotifyIconData);
    Wnd := hWindow;
    uID := ID;
    uFlags := Flags;
    hIcon := Icon;
    lstrcpyn(szTip, Tip, SizeOf(szTip));
  end;
  // Викликаємо стандартну функцію Windows Shell_NotifyIcon для зміни
  // іконки на Панелі Задач
  Result := Shell_NotifyIcon(NIM_MODIFY, @NID);
end;

// Функція для знищення іконки на Панелі Задач
function TaskBarDeleteIcon(hWindow:THandle; ID:Integer):Boolean;
var NID: TNotifyIconData;
begin
  FillChar(NID,SizeOf(TNotifyIconData),0);
  with NID do
  begin
    cbSize := SizeOf(TNotifyIconData);
    Wnd := hWindow;
    uID := ID;
  end;
  // Викликаємо стандартну функцію Windows Shell_NotifyIcon для створення
  // іконки на Панелі Задач
  Result := Shell_NotifyIcon(NIM_DELETE,@NID);
end;

end.

```

Текст модуля з константами (Constant.pas).

```

unit Constant;
(
  *****
  Цей модуль використовується як програмою, так і
  компілятором ресурсів, забезпечуючи узгоджені
  ідентифікатори для елементів меню
  *****
)
interface

const
  ID_CLOSE = 1000;
  ID_ABOUT = 1001;
  ID_DLLFORM = 1002;

implementation

end.

```

Текст файлу проекту динамічної бібліотеки (Ul.dpr).

```

// Динамічна бібліотека для показу вікна з програмки демонстрації

```

```
// використання панелі задач.
// Бібліотека використовує VCL, тому її розмір (~395 КБайт) на порядок
// більший за саму програму (її розмір ~13 КБайт).
```

```
library UI;

uses
  Forms,
  Buttons,
  UIDLL in 'UIDLL.pas' {fUI};

{$R *.RES}

exports DoDLLForm, DestroyDLLForm, ForegroundDLLForm;

begin
end.
```

Текст модуля форми діалогового вікна у динамічній бібліотеці (UIDLL.pas).

```
unit UIDLL;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls;

type
  TfUI = class(TForm)
    Label1 : TLabel;
    Edit1 : TEdit;
    BitBtn1 : TBitBtn;
    BitBtn2 : TBitBtn;
    Bevel1 : TBevel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var fUI : TfUI;

// Функція для створення вікна
function DoDLLForm(lpNewHint:PChar; iSize:Integer):Boolean; stdcall; export;
// Процедура знищення вікна
procedure DestroyDLLForm; stdcall; export;
// Процедура, яка ставить вікно на передній план
procedure ForegroundDLLForm; stdcall; export;

implementation

{$R *.DFM}

procedure ForegroundDLLForm;
begin
  if Assigned(fUI) then SetForegroundWindow(fUI.Handle);
end;

procedure DestroyDLLForm;
begin
  if Assigned(fUI) then fUI.Close;
end;

function DoDLLForm (lpNewHint:PChar; iSize:Integer):Boolean;
begin
  fUI:=TfUI.Create(nil);
  with fUI do
  try
    Edit1.Text:=lpNewHint;
    Edit1.MaxLength:=iSize-1;
```

```
    Caption:=lpNewHint;  
    SetForegroundWindow(fUI.Handle);  
    Result:=ShowModal=mrOk;  
    if Result then StrPCopy(lpNewHint, Edit1.Text);  
finally  
    if fUI<>nil then Free;  
end;  
end;  
  
end.
```

Лабораторна робота №4

Тема: Розробка зберігача екрану (Screen Saver) засобами Windows API.

Мета: Вивчення принципів побудови зберігача екрану та методів звертання до функцій Windows API. Ознайомлення з основами графіки Windows API. Узагальнення знань по Windows API.

1. Принципи написання зберігача екрану для ОС Windows

Зберігач екрану – це програма, яка активізується, якщо комп'ютер знаходиться у стані бездіяльності протягом певного заданого проміжку часу. У даному лабораторній роботі розглянуто зберігач екрану, який при активації малює на екрані прямокутники різних стилів (звичайні та скруглені) з випадковим положенням, розміром та кольором.

Зберігач екрану працює у фоновому режимі, і тому він не повинен заважати роботі інших запущених програм. Тому сам зберігач має бути як можна меншого обсягу. Для зменшення обсягу файлу в описаній нижче програмі не використовуються візуальні компоненти **Delphi**, так як включення хоча б одного з них приведе до збільшення розміру файлу понад 300 Кб, а так описана програма має розмір всього 44 Кб.

Технічно, зберігач екрану є звичайним виконавчим файлом (з розширенням **.SCR**), що керується через параметри командного рядка. Так, якщо користувач хоче змінити параметри зберігача, **Windows** виконує його з параметром **"-c"** у командному рядку. Якщо винакає необхідність показати дію зберігача у вікні попереднього перегляду, операційна система запускає зберігач з параметром **"-p"**. При потребі задання паролю на припинення виконання зберігача, він виконується з параметром **"-a"**. Тому створення зберігача екрану варто почати із процедури, яка аналізуватиме вміст командного рядка та передаватиме керування до відповідних частин програми:

```
procedure RunScreenSaver;
var S : string;
begin
  S:=ParamStr(1);
  if (Length(S)>1) then begin
    Delete(S,1,1);
    S[1]:=UpCase(S[1]);
  end;
  LoadSettings;
  if (S[1]='C') then RunSettings else // Установка параметрів зберігача екрану
  if (S[1]='P') then RunPreview else // Перегляд зберігача екрану
  if (S[1]='A') then RunSetPassword else RunFullScreen; // Встановлення паролю
end;
```

Оскільки у програмі потрібно створювати невелике вікно попереднього перегляду (яке буде відображатися у вікні "Свойства: Экран"²⁴ на сторінці "Заставка"- див. рис. 4.1) і повноекранне вікно, їх краще об'єднати, використовуючи єдиний клас вікна.

Так як зберігач екрану не повинен переставати працювати навіть при значній завантаженості процесора, для його реалізації слід використовувати окремий потік. Це має ще й ту перевагу, що не потрібно використовувати таймер.

Процедура для запуску зберігача у повноекранному режимі наступна:

```
procedure RunFullScreen;
var
  R      : TRect;
  Msg    : TMsg;
  Dummy  : DWORD;
```

²⁴ Це вікно можна відкрити з Панелі керування (Control Panel) Windows, активізувавши апплет "Экран" (Display).

```

    Foreground : hWnd;
begin
    IsPreview:=False;
    MoveCounter:=InitMoveCounter;
    Foreground:=GetForegroundWindow;
    while (ShowCursor(false)>0) do;
    GetWindowRect(GetDesktopWindow,R);
    CreateScreenSaverWindow(R.Right-R.Left,R.Bottom-R.Top,0);
    CreateThread(nil,0,@PreviewThreadProc,nil,0,Dummy);
    SystemParametersInfo(SPI_SCREENSAVERRUNNING,1,@Dummy,0);
    // Стандартний цикл обробки повідомлень
    while GetMessage(Msg,0,0,0) do begin
        TranslateMessage(Msg);
        DispatchMessage(Msg);
    end;
    SystemParametersInfo(SPI_SCREENSAVERRUNNING,0,@Dummy,0);
    ShowCursor(True);
    SetForegroundWindow(Foreground);
end;

```

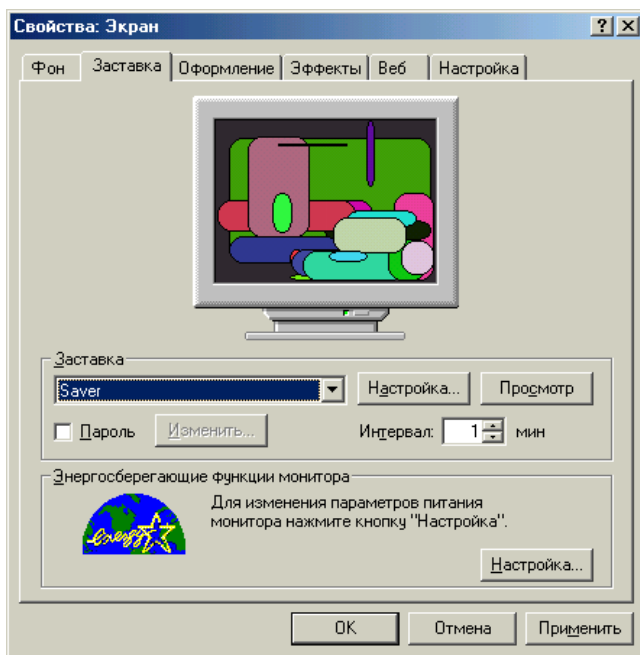


Рис 4.1. Вікно попереднього перегляду зберігача екрану.

Дана процедура ініціалізує деякі глобальні змінні (описані далі), потім ховає курсор миші і створює вікно зберігача екрану. При цьому важливо повідомляти **Windows**, що вікно належить зберігачу через функцію **SystemParametersInfo** (це відключає **Ctrl-Alt-Del**, щоб не можна було повернутися у **Windows**, не ввівши пароль). Створення вікна зберігача здійснюється функцією **CreateScreenSaverWindow**:

```

function CreateScreenSaverWindow(Width,Height : Integer; ParentWindow : hWnd) : hWnd;
var WC : TWndClass;
begin
    // Заповнення структури з інформацією про клас вікна
    with WC do begin
        Style:=cs_ParentDC;
        lpfnWndProc:=@PreviewWndProc;
        cbClsExtra:=0; cbWndExtra:=0; hIcon:=0; hCursor:=0;
        hbrBackground:=0; lpszMenuName:=nil;
        lpszClassName:='MyDelphiScreenSaverClass';
    end;
    WC.hInstance:=hInstance;

    // Реєстрація нового класу вікна
    RegisterClass(WC);
    if (ParentWindow<>0) then // Якщо вікно є вікном попереднього перегляду
        Result:=CreateWindow('MyDelphiScreenSaverClass','MySaver',

```

```

    WS_CHILD or WS_VISIBLE or WS_DISABLED,0,0,Width,Height,ParentWindow,0,hInstance,
    nil)
else begin
    // Якщо вікном є весь екран
    Result:=CreateWindow('MyDelphiScreenSaverClass','MySaver',
    WS_VISIBLE or WS_POPUP,0,0,Width,Height, 0,0,hInstance,nil);
    SetWindowPos(Result,HWND_TOPMOST,0,0,0,0,SWP_NOMOVE or SWP_NOSIZE or SWP_NOREDRAW);
end;
PreviewWindow:=Result;
end;

```

Як видно з наведеного вище коду, для створення вікна попереднього перегляду потрібно знати його ідентифікатор (**handle**). **Windows** передає його значення у командному рядку другим параметром (після "-p"). Тому процедура для попереднього перегляду має наступний вид:

```

procedure RunPreview;
var
    R          : TRect;
    PreviewWindow : hWnd;
    Msg        : TMsg;
    Dummy      : DWORD;
begin
    IsPreview:=true;
    PreviewWindow:=StrToInt(ParamStr(2));
    GetWindowRect(PreviewWindow,R);
    CreateScreenSaverWindow(R.Right-R.Left,R.Bottom-R.Top,PreviewWindow);
    CreateThread(nil,0,@PreviewThreadProc,nil,0,Dummy);
    // Стандартний цикл обробки повідомлень
    while GetMessage(Msg,0,0,0) do begin
        TranslateMessage(Msg);
        DispatchMessage(Msg);
    end;
end;

```

Ця процедура спочатку отримує ідентифікатор вікна попереднього перегляду з командного рядка і присвоює його змінній **PreviewWindow**, після цього створює вікно зберігача та його потік, який і буде виконувати роботу по малюванню у вікні зберігача. Закінчується процедура стандартним циклом обробки повідомлень, який призначений для перетворення повідомлень з черги потоку вікна та їх відправки до віконної функції, яка і здійснює їх безпосередню обробку.

Для виконання зберігача екрану використовується окремий потік, який створюється функцією **CreateThread**. Функція потоку виглядає так:

```

function PreviewThreadProc(Data : Integer) : Integer; stdcall;
var R : TRect;
begin
    Result:=0; Randomize;
    GetWindowRect(PreviewWindow,R);
    MaxX:=R.Right-R.Left;  MaxY:=R.Bottom-R.Top;
    ShowWindow(PreviewWindow,SW_SHOW);
    UpdateWindow(PreviewWindow);
    repeat
        InvalidateRect(PreviewWindow,nil,False);
        Sleep(200);
    until QuitSaver;
    PostMessage(PreviewWindow,WM_DESTROY,0,0);
end;

```

Потік змушує оновлюватися зображення у вікні зберігача (функція **InvalidateRect**), засинає на якийсь час (за допомогою функції **Sleep**), і оновлює зображення знову. А **Windows** при цьому посилає повідомлення **WM_PAINT** у вікно зберігача (не в потік). Цикл виконання потоку триває, доки змінна **QuitSaver** не матиме значення **true**. Для того, щоб обробляти це повідомлення, потрібна функція вікна:

```

function PreviewWndProc(Window : hWnd; Msg,WParam, LParam : Integer): Integer; stdcall;
begin

```

```

Result := 0;
case Msg of
  WM_NCCREATE : Result:=1;
  WM_DESTROY : PostQuitMessage(0);
  WM_PAINT    : DrawSingleBox;
  WM_KEYDOWN  : QuitSaver:=AskPassword;
  WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN, WM_MOUSEMOVE :
    if (not IsPreview) then begin
      Dec(MoveCounter);
      if (MoveCounter<=0) then QuitSaver:=AskPassword;
    end;
  else Result:=DefWindowProc(Window,Msg,WParam,LParam);
end;
end;
end;

```

Віконна процедура обробляє повідомлення, список та призначення яких показано у таблиці №4.1.

Таблиця №4.1

Повідомлення	Призначення
WM_NCCREATE	Посилається перед створенням вікна. Якщо прикладна програма обробляє його, вона може повернути TRUE для продовження створення вікна, або FALSE для припинення.
WM_DESTROY	Посилається, коли вікно буде знищуватись.
WM_PAINT	Посилається, коли є потреба перемалювати вміст вікна.
WM_KEYDOWN	Виникає при натисканні на клавішу клавіатури.
WM_LBUTTONDOWN	Виникає при натисканні на ліву кнопку миші.
WM_MBUTTONDOWN	Виникає при натисканні на середню кнопку миші.
WM_RBUTTONDOWN	Виникає при натисканні на праву кнопку миші.
WM_MOUSEMOVE	Виникає при русі миші.

Якщо відбувся рух миші, або натиснено її кнопку чи клавішу клавіатури, робота зберігача екрану припиняється. Проте перед цим слід перепитати в користувача пароль:

```

function AskPassword : Boolean;
var
  Key    : hKey;
  D1,D2 : Integer;
  Value  : Integer;
  Lib    : THandle;
  F      : TVSSPFunc;
begin
  Result:=true;
  if (RegOpenKeyEx(HKEY_CURRENT_USER,'Control Panel\Desktop',0,
  KEY_READ,Key)=ERROR_SUCCESS) then
  begin
    D2:=SizeOf(Value);
    // Перевірка, чи використовується пароль для зберігача екрану
    if (RegQueryValueEx(Key, 'ScreenSaveUsePassword',nil,@D1,@Value,@D2)=
    ERROR_SUCCESS) then
    begin
      if (Value<>0) then begin
        Lib:=LoadLibrary('PASSWORD.CPL');
        if (Lib>32) then begin
          @F:=GetProcAddress(Lib,'VerifyScreenSavePwd');
          ShowCursor(True);
          if (@F<>nil) then Result:=F(PreviewWindow);
          ShowCursor(False);
          MoveCounter:=InitMoveCounter;
          FreeLibrary(Lib);
        end;
      end;
    end;
  end;
  RegCloseKey(Key);
end;
end;

```


Функція **AskPassword** спочатку зчитує з реєстру значення **ScreenSaveUsePassword** для перевірки необхідності використання пароллю. Якщо пароль встановлено, то завантажується системна бібліотека **PASSWORD.CPL**, яка містить функцію **VerifyScreenSavePwd** для перевірки пароля зберігача. Після отримання адреси точки входу у вказану функцію (**GetProcAddress**) здійснюється перепитування пароллю. В кінці бібліотека **PASSWORD.CPL** вивільняється з пам'яті функцією **FreeLibrary**.

Тип функції для перевірки пароллю зберігача **TVSSFunc** стандартом **Windows** визначений так:

```
type TVSSPFunc = function(Parent : hWnd) : Bool; stdcall;
```

Як параметр функції передається ідентифікатор вікна зберігача екрану. Якщо пароль введено вірно, функція повертає значення **true**, в іншому випадку – **false**. Отримане значення передається у результат функції **AskPassword**.

Для показу діалогового вікна конфігурації зберігача використовується процедура **RunSettings**:

```
procedure RunSettings;
var SSDialog : Integer;
begin
  SSDialog:=DialogBox(hInstance,'SaverSettingsDlg',0,@SettingsDlgProc);
  // Якщо було натиснено кнопку "ОК", то зберігаються настройки
  if (SSDialog=idOK) then SaveSettings;
end;
```

Тепер потрібно створити шаблон для діалогового вікна **SaverSettingsDlg** (стандартні форми **Delphi** не використовуємо). Сценарій для діалогу зміни параметрів наступний²⁵:

```
SaverSettingsDlg DIALOG 70, 130, 200, 60 STYLE WS_POPUP | WS_DLGFRAAME | WS_SYSMENU
CAPTION "Настройки для SSaver"
FONT 8, "MS Sans Serif"
BEGIN
  LTEXT "Тип прямокутників:", 1, 4, 3, 100, 9
  DEFPUSHBUTTON "ОК", 5, 150, 11, 46, 16
  PUSHBUTTON "Відміна", 6, 150, 36, 46, 16
  LTEXT "Демо 'SSaver'. ТДТУ, ICSoft©, 2002.", 7,4,45,130,16, WS_CHILD | WS_VISIBLE
END
```

Призначення використаних елементів синтаксису показано в таблиці №4.2.

Таблиця №4.2

Ключ	Призначення
DIALOG	Вказує, що створюється діалогове вікно. Перед ключем ставиться назва шаблону діалогового вікна, а після- положення вікна, його розміри та стиль.
STYLE	Задає стиль вікна.
WS_POPUP	Звичайне вікно.
WS_DLGFRAAME	Вікно має подвійну рамку.
WS_SYSMENU	Вікно має системне меню в заголовку.
WS_CHILD	Вказує, що вікно є дочірнім.
WS_VISIBLE	Вказує, що вікно є нормально видимим.
CAPTION	Заголовок вікна.
FONT	Шрифт вікна. Після ключа вказується розмір та назва шрифту.
BEGIN	Початок блоку, який визначає елементи вікна.
END	Кінець блоку, який визначає елементи вікна.
LTEXT	Елемент-аналог компонента TLabel з VCL Delphi. Після ключа вказується текст компонента, його ідентифікатор, відступ від лівого краю вікна, відступ зверху вікна, ширина та висота.
DEFPUSHBUTTON	Кнопка, яка використовується по замовчужанню. Аналог компонента TButton з VCL Delphi. Після ключа вказується текст у кнопці, її ідентифікатор, відступ від лівого краю вікна, відступ зверху вікна, ширина та висота ²⁶ .

²⁵ Для створення таких сценаріїв використовується мова сценаріїв ресурсів (Resource Script Language).

²⁶ Наприклад, для кнопки "ОК" ідентифікатор дорівнює 5, відступ від лівого краю вікна 150, від верхнього краю 11, ширина кнопки 46 і висота 16.

Файл сценарію зберігається як звичайний текстовий файл²⁷, і компілюється, наприклад, з використанням компілятора ресурсів **BRCC32.EXE** фірми **Borland**, який поставляється разом з **Delphi**. Якщо файл сценарію назвати **SETTINGS.RC**, то в результаті його компіляції отримається файл ресурсів **SETTINGS.RES**, який міститиме шаблон розробленого діалогу. Для під'єднання цього ресурсного файлу до проекту **Delphi** використовується така директива компілятора²⁸:

```
{ $R SETTINGS.RES }
```

Функція вікна для діалогу налаштувань наступна:

```
function SettingsDlgProc(Window : hWnd; Msg,WParam,LParam : Integer): Integer; stdcall;
begin
  Result:=0;
  case Msg of
    WM_INITDIALOG :
      begin
        // Ініціалізація діалогового вікна
        Result:=0;
        hcbType:=CreateWindow('Combobox','cbType',WS_CHILD or CBS_DROPDOWNLIST or
          CBS_HASSTRINGS,6,20,200,80, Window ,0,hInstance,nil);
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Звичайні прямокутники')));
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Скруглені прямокутники')));
        ShowWindow(hcbType,SW_SHOW);
        SendMessage(hcbType,CB_SETCURSEL,DWORD(RoundedRectangles),0);
      end;
    WM_COMMAND : if (LoWord(WParam)=5) then begin // Натиснено кнопку "ОК"
        RoundedRectangles:=Boolean(SendMessage(hcbType,CB_GETCURSEL,0,0));
        EndDialog(Window,idOK)
      end
        // Натиснено кнопку "Відміна"
      else if (LoWord(WParam)=6) then EndDialog(Window,idCancel);
    WM_CLOSE : DestroyWindow(Window);
    WM_DESTROY : PostQuitMessage(0);
  else Result:=0;
  end;
end;
```

Вигляд розробленого діалогового вікна показано на рис. 4.2.

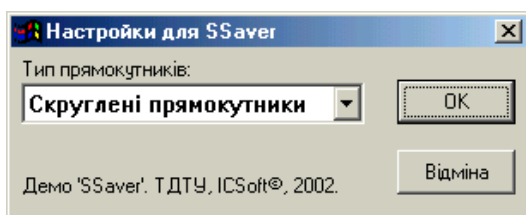


Рис. 4.2. Діалогове вікно налаштувань.

Розроблений зберігач екрану в активному стані малює на екрані різнокольорові прямокутники- звичайні або скруглені. При ініціалізації вікна (тобто при отриманні повідомлення **WM_INITDIALOG**) створюється ще один елемент вікна – випадаючий список (**ComboBox**), з якого користувач може вибирати параметр зберігача екрану – стиль прямокутників (звичайні або скруглені). Створення випадаючого списку здійснюється за допомогою функції **CreateWindow**. Нові рядки до випадаючого списку додаються через посилання до його вікна повідомлення **CB_ADDSTRING** з другим параметром, який містить вказівник на рядок, що треба додати. Активний елемент у випадаючому списку визначається змінною **RoundedRectangles** і встановлюється шляхом відправлення його вікну повідомлення **CB_SETCURSEL**, перший параметр якого вказує на порядковий номер елемента, що слід зробити активним (0 – перший елемент, 1 – другий...).

²⁷ Для цього можна використати стандартний текстовий редактор "Блокнот" (**Notepad**) **Windows**.

²⁸ Її можна помістити відразу після директиви, яку **Delphi** додає у проект автоматично- `{ $R *.res }`, і яка під'єднує файл ресурсів самого проекту (його назва співпадає з назвою проекту).

Крім цього, функція вікна обробляє повідомлення **WM_COMMAND**, яке отримується при натисканні на кнопки "ОК" та "Відміна" у діалоговому вікні. При цьому нижнє слово першого параметра **WParam** містить ідентифікатор елемента вікна (кнопки), яку було натиснено. Так, якщо натискається кнопка "ОК", нижнє слово змінної **WParam** дорівнює 5 (див. текст сценарію діалогового вікна), а при натисканні кнопки "Відміна" - 6. При натисканні на кнопку "ОК" визначається номер активного елемента у списку стилю прямокутників (це робиться через повідомлення **CB_GETCURRESEL**, яке і повертає цей номер) і задається відповідне значення змінній **RoundedRectangles**. Закривається діалогове вікно через виклик функції **EndDialog**.

Після того, як користувач задав деякі параметри, потрібно їх зберегти. Це здійснює процедура **SaveSettings**.

```
procedure SaveSettings;
var
  Key    : hKey;
  Dummy  : Integer;
begin
  if (RegCreateKeyEx(HKEY_CURRENT_USER, 'Software\ICSoft\SSaver', 0, nil, REG_OPTION_NON_VOLATILE,
    KEY_ALL_ACCESS, nil, Key, @Dummy) = ERROR_SUCCESS) then begin
    RegSetValueEx(Key, 'RoundedRectangles', 0, REG_BINARY, @RoundedRectangles, SizeOf(Boolean));
    RegCloseKey(Key);
  end;
end;
```

Зберігається значення змінної **RoundedRectangles** у реєстрі **Windows**, в розділі **HKEY_CURRENT_USER**, підрозділ **'Software\ICSoft\SSaver'**. Назва параметра **'RoundedRectangles'** співпадає з назвою відповідної змінної. Для зберігання значення у реєстрі спочатку за допомогою функції **RegCreateKeyEx** створюється (або відкривається, якщо вже існує) відповідний підрозділ реєстру. Після встановлення відповідного значення за допомогою функції **RegSetValueEx** розділ реєстру закривається функцією **RegCloseKey**.

Завантажує параметри з реєстру процедура **LoadSettings**.

```
procedure LoadSettings;
var
  Key    : hKey;
  D1,D2  : Integer;
  Value  : Boolean;
begin
  if (RegOpenKeyEx(HKEY_CURRENT_USER, 'Software\ICSoft\SSaver', 0, KEY_READ, Key) = ERROR_SUCCESS)
  then begin
    D2 := SizeOf(Value);
    if (RegQueryValueEx(Key, 'RoundedRectangles', nil, @D1, @Value, @D2) = ERROR_SUCCESS) then
      RoundedRectangles := Value;
    RegCloseKey(Key);
  end;
end;
```

Спочатку за допомогою функції **RegOpenKeyEx** робиться спроба відкрити підрозділ реєстру **'HKEY_CURRENT_USER\Software\ICSoft\SSaver'**, у якому міститься параметр нашого зберігача екрану. Якщо відповідний підрозділ вдалось відкрити, то функція **RegQueryValueEx** читає значення з реєстру під назвою **'RoundedRectangles'** та присвоює його однойменній змінній у програмі. Після цього відкритий розділ реєстру закривається та вивільняється пам'ять, яка була використана на зберігання його ідентифікатора та інших пов'язаних із ним даних.

Для установки пароля використовується процедура **RunSetPassword**.

```
procedure RunSetPassword;
var
  Lib : THandle;
  F   : TPCPAFunc;
begin
  Lib := LoadLibrary('MPR.DLL');
  if (Lib > 32) then begin
```

```

@F:=GetProcAddress(Lib,'PwdChangePasswordA');
if (@F<>nil) then F('SCRSAVE',StrToInt(ParamStr(2)),0,0);
FreeLibrary(Lib);
end;
end;

```

Спочатку динамічно завантажується (недокументована) бібліотека **MPR.DLL**, яка містить функцію для установки пароля зберігача екрану.

Тип функції для установки паролю **TPCPAFunc**²⁹ стандартом Windows визначений так:

```

type TPCPAFunc = function(A : PChar; Parent : hWnd; B,C : integer) : integer; stdcall;

```

Як параметр функції передається ідентифікатор вікна зберігача (**Parent**), який є другим командним параметром при виклику зберігача екрану. Параметр **A** повинен містити значення **'SCRSAVE'**, яке вказує, що пароль встановлюється для зберігача екрану. Параметри **B** та **C** недокументовані (можуть бути просто нулі).

Тепер розглянемо створення графіки. Для прикладу, реалізовано процедуру малювання на екрані прямокутників заданого стилю та випадкового розміру, положення і кольору. Цю операцію здійснює процедура **DrawSingleBox**.

```

procedure DrawSingleBox;
var
  PaintDC : HDC;
  Info : TPaintStruct;
  OldBrush : hBrush;
  X,Y : Integer;
  Color : LongInt;
begin
  PaintDC:=BeginPaint(PreviewWindow,Info);
  X:=Random(MaxX); Y:=Random(MaxY);
  Color:=RGB(Random(255),Random(255),Random(255));
  OldBrush:=SelectObject(PaintDC,CreateSolidBrush(Color));
  if RoundedRectangles then RoundRect(PaintDC,X,Y,X+Random(MaxX-X),Y+Random(MaxY-Y),20,20)
  else Rectangle(PaintDC,X,Y,X+Random(MaxX-X),Y+Random(MaxY-Y));
  DeleteObject(SelectObject(PaintDC,OldBrush));
  EndPaint(PreviewWindow,Info);
end;

```

Підготовка вікна зберігача до малювання здійснюється функцією **BeginPaint**, якій передається ідентифікатор вікна зберігача. У відповідності до значення змінної **RoundedRectangles** малюються або звичайні прямокутники (для цього використовується функція **Rectangle**), або скруглені прямокутники (за допомогою функції **RoundRect**).

Тепер розглянемо глобальні змінні програми (табл. №4.3):

Таблиця №4.3

Змінна	Тип	Призначення
IsPreview	boolean	Показує, чи вікно зберігача є вікном попереднього перегляду (true), чи повноекранним (false).
MoveCounter	integer	Змінна, яка визначає лічильник подій, обнулення якого приведе до закінчення роботи зберігача екрану. У програмі їй присвоюється початкове значення InitMoveCounter = 3. Такими подіями є натискання кнопки миші, рух миші, натискання клавіші клавіатури.
QuitSaver	boolean	Вказує, чи припиняти роботу зберігача (true).
PreviewWindow	hWnd	Ідентифікатор вікна зберігача екрану.
MaxX, MaxY	integer	Розміри вікна зберігача.
RoundedRectangles	boolean	Параметр, який вказує на стиль прямокутників, які буде малювати зберігач- звичайні (false) чи скруглені (true).
hcbType	HWND	Ідентифікатор випадального списку (ComboBox), яким задається параметр RoundedRectangles у діалоговому вікні налаштувань зберігача.

²⁹ Дана функція недокументована Microsoft.

І під кінець слід описати основний програмний блок у файлі проекту, який і викликає роботу зберігача екрану. Він містить виклик лише однієї процедури – **RunScreenSaver**.

```
begin
  RunScreenSaver;
end.
```

Для досягнення максимальної швидкодії та мінімального розміру виконавчого файлу, при розробці даного зберігача екрану використовувались лише функції **Windows API**. Проте у певній мірі роботу можна спростити, якщо вікно задання параметрів створити за допомогою засобів **Delphi** та скомпілювати у окрему динамічну бібліотеку. Тоді її при необхідності можна завантажувати, і після введення параметрів вивільняти з пам'яті. Так як параметри вводяться одноразово на порівняно тривалий період, то швидкодія самого зберігача екрану від цього не буде суттєво падати. Таким чином може бути поєднано швидкодіючий та економний щодо споживання ресурсів комп'ютера код **Windows API** та громіздкий (проте ефективний та економічний з точки зору швидкості розробки) код **VCL Delphi**.

2. Завдання на лабораторну роботу

Написати програму зберігача екрану з використанням функцій **Windows API**, без впровадження компонентів **VCL Delphi**. Програма повинна забезпечувати показ, попередній перегляд, ввід параметрів та пароля для розробленого зберігача. Тип дії зберігача та параметри, зміну яких необхідно передбачити у програмі, вказано у таблиці №4.4.

Таблиця №4.4

Варіант	Видима дія зберігача на екрані	Регульовані параметри
1	Кулька, яка "літає" по екрані, відбиваючись від його країв	Розмір та швидкість руху кульки
2	Промінь, який починається в центрі екрану, йде до його краю та обертається подібно до стрілки годинника	Напрямок та швидкість обертання променя
3	Кулька, яка рухається по спіралі, що починається з центра екрану і слідує до його країв	Колір кульки та крок між витками спіралі
4	Прямокутник, який рухається по екрані, відбиваючись від його країв	Колір та швидкість руху прямокутника
5	Текстова фраза, яка рухається по екрані	Текст фрази та швидкість її руху
6	Кулька, яка з'являється на екрані, зникає, потім з'являється в іншому місці з іншим кольором, і т.д.	Розмір кульки та тривалість її показу
7	Прямокутник, який рухається по спіралі, що починається з центра екрану і слідує до його країв	Швидкість та напрям руху прямокутника
8	На екрані розташоване коло, по якому рухається круг	Колір круга та швидкість його руху
9	На екрані в різних місцях з'являються окремі літери із заданої текстової фрази	Текст фрази та розмір літер
10	На екрані розташоване коло, по якому рухається круг	Розмір та положення кола
11	Дві кульки, які виникають з точок, збільшуються до певного розміру, потім поступово зменшуються назад у точку і так пульсують	Розміри кульок
12	Прямокутник, який з'являється на екрані, зникає, потім з'являється в іншому місці з іншим розміром, і т.д.	Розміри прямокутника
13	Текстова фраза, яка з'являється на екрані, показується протягом заданого часу, зникає, з'являється в іншому місці і т.д.	Текст фрази і розмір шрифту
14	Круг, який обертається навколо своєї осі та з кожним оборотом змінює колір	Розмір круга та швидкість обертання
15	Круги, які з'являються на екрані у випадкових положеннях та з випадковими кольорами	Розмір та частота показу кругів
16	Квадрат, який обертається навколо своєї осі та з кожним оборотом змінює колір	Положення квадрата на екрані
17	Лінії випадкового кольору, які з'являються на екрані у випадкових положеннях	Товщина лінії та її тип (суцільна, пунктирна...)
18	Правильний шестикутник, який рухається по екрані, відбиваючись від його країв	Розмір шестикутника, колір лінії
19	Фігура у вигляді сонця (☀), яка рухається по екрані	Колір "сонця" та швидкість руху

Варіант	Видима дія зберігача на екрані	Регульовані параметри
20	Кола, еліпси, прямокутники та лінії, які випадковим чином з'являються у випадкових місцях з випадковим кольором	Частота появи фігур та один їх лінійний розмір
21	Лінія, яка рухається по екрану, міняючи своє положення, нахил та швидкість	Швидкість лінійного та швидкість обертового руху лінії
22	Різні геометричні фігури, які виникають з центру екрану і рухаються по спіралі до країв	Швидкість руху та крок спіралі
23	Сніжинки, що з'являються в різних місцях екрану і через кожен заданий інтервал часу змінюють колір	Розмір сніжинок та інтервал часу, коли міняється колір
24	Два круга, які випадковим чином рухаються по екрані	Розмір одного та розмір другого кола
25	Прямокутник, який рухається по екрані, відбиваючись від країв і змінюючи колір при кожному зіткненні з краєм екрану	Розміри прямокутника (ширина і висота)
26	Різні геометричні фігури різних розмірів, які з'являються випадковим чином в різних місцях екрану	Розміри та частота появи фігур
27	Шестикутник, який з'являється на екрані, зникає, потім з'являється в іншому місці з іншим розміром, і т.д.	Частота появи шестикутника і максимальний розмір
28	Прямокутник, який рухається по екрані, змінюючи розміри	Ширина і висота прямокутника
29	Трикутник, який рухається по спіралі з середини екрану і змінює розмір та форму	Швидкість руху трикутника і крок спіралі
30	На екрані по коловій траєкторії рухаються різні геометричні фігури, тип фігури змінюється з кожним оборотом	Розмір та положення колової траєкторії

Лабораторна робота №5

Тема: Отримання інформації про операційну систему та ресурси комп'ютера засобами Windows API.

Мета: Вивчення функцій Windows API, призначених для отримання найважливішої інформації про операційну систему та характеристики обладнання.

1. Отримання інформації про ОС та встановлене обладнання

Ряд програм для своєї коректної роботи можуть потребувати відомостей про платформу ОС³⁰ або характеристики встановленого обладнання. Так, різні платформи ОС мають дещо інший набір функцій API, і виклик функції, характерної лише для однієї платформи (наприклад, **Win NT**) з-під іншої платформи (наприклад, **Win 9x**), приведе до виникнення помилки. **Windows API** містить ряд функцій, призначених для отримання інформації як про саму операційну систему, так і про встановлене обладнання (пам'ять, процесор, вінчестери та інше).

Текст програми для визначення основних параметрів ОС та обладнання ЕОМ приведено у пункті 3. У даній програмі для отримання інформації про ОС застосовуються функції API, проте відображення отриманих даних здійснюється за допомогою форми, створеної через **VCL Delphi**. Головне вікно програми **SysInfo** показано на рис. 5.1.

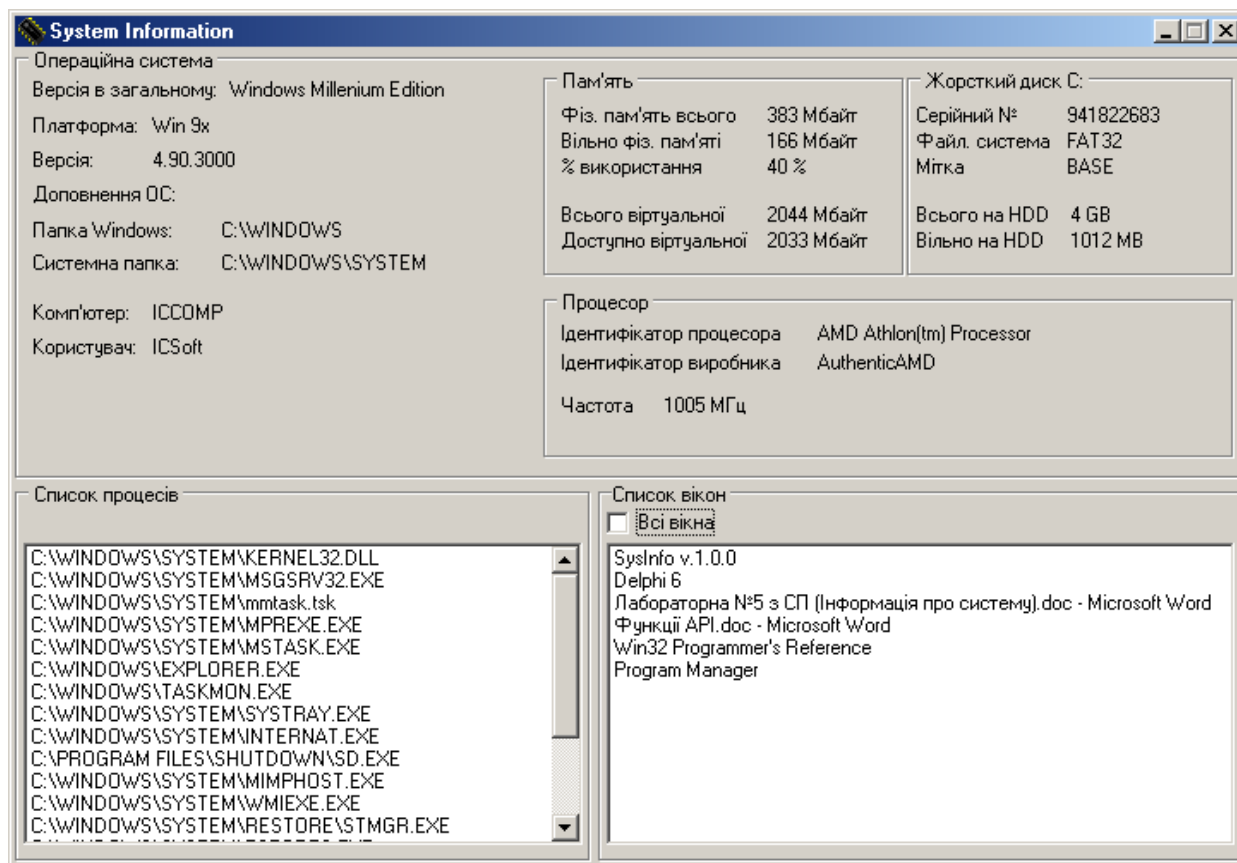


Рис. 5.1. Головне вікно програми SysInfo.

³⁰ ОС – операційна система.

1.1. Отримання загальної інформації про ОС

Для визначення основних характеристик операційної системи **Windows** використовується функція **GetVersionEx**. У дану функцію передають один параметр – вказівник на структуру³¹ типу **TOSVersionInfo**, в яку і поміщується вся отримана інформація. Елемент **dwOSVersionInfoSize** даної структури повинен містити її розмір, тому він заповнюється до виклику функції **GetVersionEx**.

Процедура для отримання основної інформації про систему **GetMainSysInfo** може мати наступний вид:

```
var
  Version      : TOSVersionInfo;
  .....

procedure TfMain.GetMainSysInfo;
var VW : integer;
begin
  with Version do begin
    dwOSVersionInfoSize:=SizeOf(TOSVersionInfo);
    GetVersionEx(Version);
    case dwPlatformId of
      VER_PLATFORM_WIN32_WINDOWS : begin lPlatform.Caption:='Win 9x';
                                      Build:=IntToStr(LoWord(dwBuildNumber));
                                      end;
      VER_PLATFORM_WIN32_NT      : begin lPlatform.Caption:='Win NT';
                                      Build:=IntToStr(dwBuildNumber); end;
    end;
    lVersionNum.Caption:=IntToStr(dwMajorVersion)+'.'+IntToStr(dwMinorVersion)+'.'+Build;
    lExt.Caption:=szCSDVersion;
  end;

  CNSize:=200;
  if GetUserName(CN, CNSize) then lUser.Caption:=CN;

  CNSize:=200;
  if GetComputerName(CN, CNSize) then lCompName.Caption:=CN;

  CNSize:=200;
  if Boolean(GetSystemDirectory(CN, CNSize)) then lSysDir.Caption:=CN;

  CNSize:=200;
  if Boolean(GetWindowsDirectory(CN, CNSize)) then lWinDir.Caption:=CN;

  VW:=GetWindowsVersion;
  case VW of
    cOSUnknown : lVerInfo.Caption:='Unknown Windows';
    cOSWin95    : lVerInfo.Caption:='Windows 95';
    cOSWin98    : lVerInfo.Caption:='Windows 98';
    cOSWin98SE  : lVerInfo.Caption:='Windows 98 SE';
    cOSWinME    : lVerInfo.Caption:='Windows Millenium Edition';
    cOSWinNT    : lVerInfo.Caption:='Windows NT';
    cOSWin2000  : lVerInfo.Caption:='Windows 2000';
    cOSWinXP    : lVerInfo.Caption:='Windows XP';
  end;
end;
```

При цьому **CN** – змінна типу **PChar** (рядок з кінцевим нулем) – ініціалізується до виклику даної процедури. **CNSize** – змінна цілого типу, яка містить розмір рядка **CN**. Так, для виділення пам'яті 200 байтів для даного рядка можна використати наступний код:

```
CNSize:=200; GetMem(CN,CNSize);
```

Після використання пам'яті, виділену під дану змінну, слід вивільнити:

³¹ Тип даних "структура" у даному випадку відповідає типу даних "запис" (record), прийнятому в мові Object Pascal.

FreeMem(CN);

В результаті виклику функції **GetVersionEx** можна отримати таку інформацію про ОС: номер версії, номер збірки ОС, ідентифікатор платформи (**Win 32s**, **Win 9x**, **Win NT**), та відомості про доповнення ОС (наприклад, **Service Pack 5**).

На основі даних про платформу ОС та мінорний і мажорний номер її версії можна зробити висновок про тип інсталюваної ОС (**Windows 95**, **Windows 98**, **Windows NT** чи інша). Для здійснення цього призначена функція **GetWindowsVersion**. Блок-схему алгоритму визначення типу **Windows** приведено на рис. 5.2.

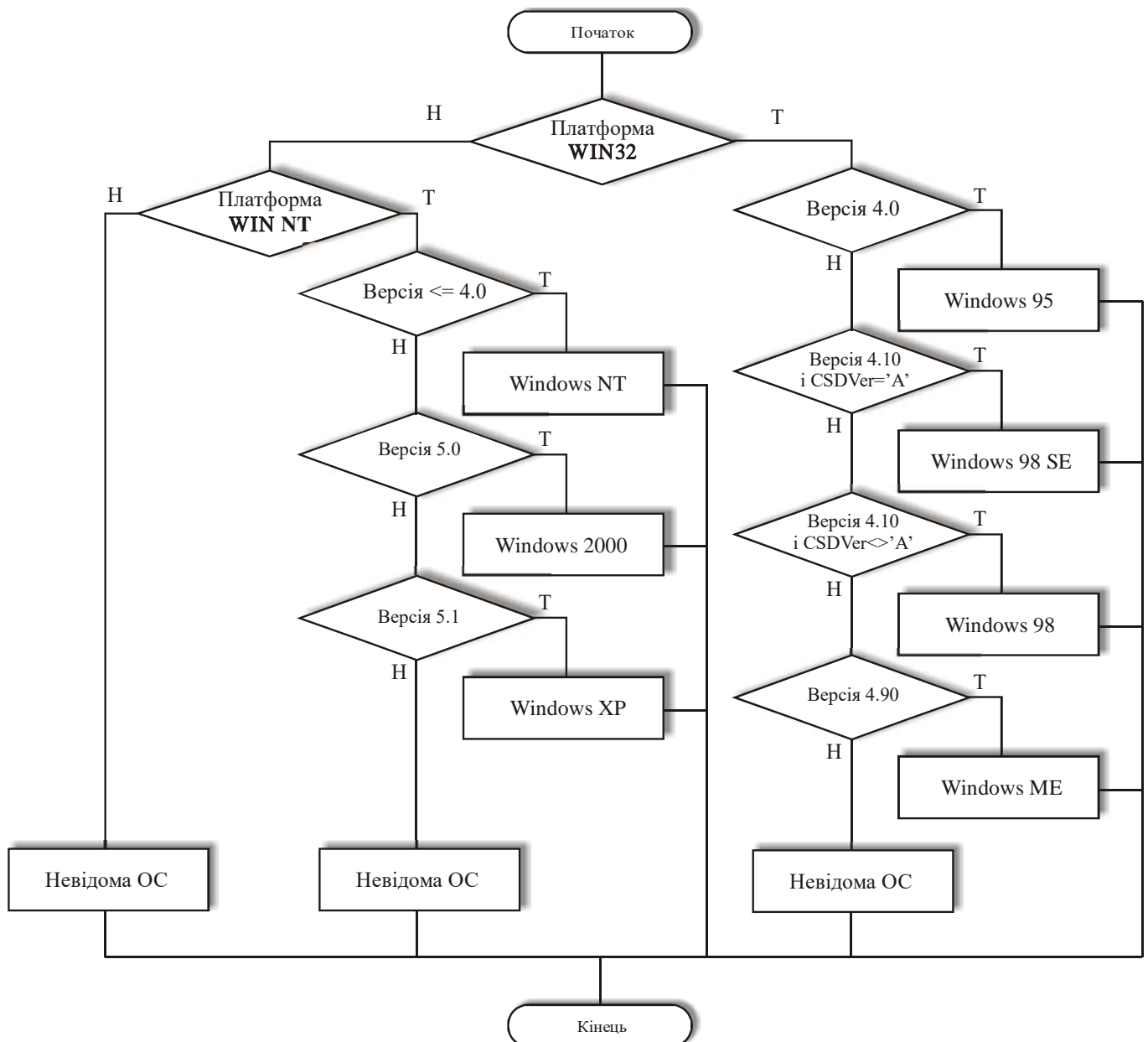


Рис. 5.2. Блок-схема алгоритму визначення типу ОС **Windows**.

Якщо операційна система підтримує багатокористувацький режим роботи, то назву активного користувача можна отримати за допомогою функції **GetUserName**. Мережеву назву комп'ютера, на якому виконується програма, повертає функція **GetComputerName**.

Назву папки, в яку інсталювано операційну систему **Windows**, можна отримати через функцію **GetWindowsDirectory**. Назву системної папки визначає функція API **GetSystemDirectory**.

1.2. Отримання інформації про пам'ять

Отримати інформацію про кількість пам'яті та ступінь її використання можна за допомогою функції **Windows API GlobalMemoryStatus**. Як параметр їй передається структура, у якій будуть збережені дані про пам'ять. Ця функція повертає дані про загальну та доступну кількість фізичної і віртуальної пам'яті, а також про стан файлу підкачки та ступінь використання пам'яті. Метод **GetMemoryInfo** для визначення вказаної інформації про пам'ять приведено у лістингу в пункті 3.

1.3. Отримання інформації про дискові накопичувачі

Метод **GetHDInfo** класу **TfMain** для отримання необхідних даних про логічний диск, з якого запускається програма, використовує функції API **GetVolumeInformation** (для визначення даних про файлову систему і параметри тому) та **GetDiskFreeSpaceEx** (для визначення розміру тому та кількості вільного місця на ньому). Цей метод має такий вигляд:

```
procedure TfMain.GetHDInfo;
var VolumeName,
    FileSystemName      : array [0..MAX_PATH-1] of Char;
    VolumeSerialNo      : DWORD;
    MaxComponentLength,
    FileSystemFlags     : DWORD;
    SC                  : PChar;
    MainDir             : string;
    FreeAvailable,
    TotalSpace          : TLargeInteger;
    TotalFree           : TLargeInteger;
begin
    MainDir:=ExtractFilePath(Application.ExeName);
    gbHD.Caption:=' Жорсткий диск '+MainDir[1]+MainDir[2]+' ';
    SC:=StrAlloc(4);
    StrPCopy(SC, MainDir[1]+MainDir[2]+MainDir[3]);
    GetVolumeInformation(SC, VolumeName, MAX_PATH, @VolumeSerialNo,
        MaxComponentLength, FileSystemFlags, FileSystemName, MAX_PATH);
    LVNum.Caption:=IntToStr(VolumeSerialNo);
    lFileSys.Caption:=FileSystemName;
    lLabel.Caption:=VolumeName;

    GetDiskFreeSpaceEx(SC, FreeAvailable, TotalSpace, @TotalFree);
    lTotalSpace.Caption:=IntToStr(TotalSpace div (1024*1024*1024))+' GB';
    lTotalFree.Caption:=IntToStr(TotalFree div (1024*1024))+' MB';
    StrDispose(SC);
end;
```

Змінній **MainDir** (типу **string**) за допомогою методу **ExtractFilePath** класу **TApplication** присвоюється назва каталогу, з якого запускається програма. Змінна **SC** (яка є рядком з кінцевим нулем – **PChar**) отримує назву диска із рядка **MainDir**. Функція **GetVolumeInformation** дозволяє визначити назву файлової системи (наприклад, **FAT** чи **NTFS**), серійний номер тому (він створюється при форматуванні диска і не є серійним номером, що призначається виробником вінчестера), а також назву вказаного тому. Для визначення обсягу вільного та загально доступного місця у заданому томі використовується функція **GetDiskFreeSpaceEx**.

Після визначення перелічених параметрів вони відображаються за допомогою компонентів типу **TLabel** на екрані.

1.4. Отримання даних про центральний процесор

Метод **GetCPUInfo** класу **TfMain** використовується для збирання інформації про тип мікропроцесора (МП), його виробника та визначення тактової частоти МП. В повному обсязі текст цього методу приведено у лістингу в пункті 3. Дані про ідентифікатор процесора та виробника

беруться з реєстру **Windows** (а саме, з вітки **HARDWARE\DESCRIPTION\System\CentralProcessor\0**), яка розташована в розділі **HKEY_LOCAL_MACHINE**). При цьому значення реєстру **'Identifier'** містить дані про мікропроцесор, а значення **'VendorIdentifier'** – про його виробника.

Для визначення тактової частоти процесора використовується процедура **GetCPUSpeed**, основний виконавчий блок якої написаний на асемблері.

```
function GetCPUSpeed: Double;
const DelayTime = 500;
var TimerHi : DWORD;
    TimerLo : DWORD;
    PriorityClass : Integer;
    Priority : Integer;
begin
    PriorityClass := GetPriorityClass(GetCurrentProcess);
    Priority := GetThreadPriority(GetCurrentThread);
    SetPriorityClass(GetCurrentProcess, REALTIME_PRIORITY_CLASS);
    SetThreadPriority(GetCurrentThread, THREAD_PRIORITY_TIME_CRITICAL);
asm
    push 10
    call Sleep
    DW 310Fh
    MOV TimerLo, EAX
    MOV TimerHi, EDX
    push DelayTime
    call Sleep
    DW 310Fh
    SUB EAX, TimerLo
    SBB EDX, TimerHi
    MOV TimerLo, EAX
    MOV TimerHi, EDX
end;
SetThreadPriority(GetCurrentThread, Priority);
SetPriorityClass(GetCurrentProcess, PriorityClass);
Result := TimerLo / (1000.0 * DelayTime);
end;
```

Код для визначення тактової частоти процесора наданий фірмою **Intel**, і оснований на внутрішніх особливостях роботи мікропроцесорів з ядром **x86**. Спочатку поточному процесу (тобто процесу даної програми) встановлюється найвищий пріорітет- пріорітет реального часу (**REALTIME_PRIORITY_CLASS**), а активному потоку встановлюється максимальне значення пріорітету (**THREAD_PRIORITY_TIME_CRITICAL**). При цьому інші процеси (в тому числі процеси самої операційної системи) перестають виконуватися. Далі за допомогою функції **Sleep** на час **DelayTime** призупиняється виконання активного потоку, і аналізується стан регістрів **EAX** та **EDX** мікропроцесора.

Після проведення обчислень активному процесу та потоку встановлюються попередньо збережені значення пріоритетів.

1.5. Отримання списку процесів

Для отримання списку активних на даний момент процесів використовується метод **GetProcessInfo** класу **TfMain**:

```
procedure TfMain.GetProcessInfo;
begin
    pe.dwSize:=SizeOf(pe);
    hSnap:=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
    if Process32First(hSnap,pe) then
    begin
        lbProcesses.Items.Add(pe.szExeFile);
        while Process32Next(hSnap,pe) do lbProcesses.Items.Add(pe.szExeFile);
    end;
end;
```

Пошук процесів здійснюється за допомогою процедур **Process32First** (шукає перший процес) та **Process32Next** (шукає всі наступні), які зберігають дані про активні процеси у змінній **pe** типу **TProcessEntry32**. Опис цих процедур здійснюється у модулі **TLHelp32**, тому він повинен бути поміщений у секцію **uses** головного модуля програми. Назви виконавчих файлів, асоційованих із знайденими процесами, поміщуються у список компонента **lbProcesses** типу **TListBox**.

1.6. Отримання списку вікон у системі

Для отримання списку вікон у системі використовується функція **Windows API EnumWindows**. Вона перелічує всі вікна верхнього рівня у системі, та ідентифікатор кожного знайденого вікна передає у програмно задану функцію зворотнього виклику **EnumWndFunc**. Сам метод **GetWindowsInfo** класу **TfMain**, який і призначений для отримання списку вікон, містить лише один виклик функції **EnumWindows**:

```
procedure TfMain.GetWindowsInfo;
begin
  EnumWindows(@EnumWndFunc, 0);
end;
```

Всю ж корисну роботу (у нашому випадку це створення списку вікон) здійснює функція **EnumWndFunc**:

```
function EnumWndFunc(Hnd:HWND;PrID:DWORD):boolean; stdcall;
var lpS : LPSTR;
begin
  Result:=true;
  GetMem(lpS,127);
  if fMain.cbAllWindows.Checked then begin
    if (GetWindowText(Hnd,lpS,127)<>0) then if lpS<>' ' then fMain.lbWindows.Items.Add(lpS);
  end else
    if (IsWindowVisible(Hnd)) and (GetWindow(Hnd,GW_OWNER)=0) and
      (GetWindowText(Hnd,lpS,127)<>0) then if lpS<>' ' then fMain.lbWindows.Items.Add(lpS);
  FreeMem(lpS,127);
end;
```

За допомогою функції **GetWindowText** отримується заголовок вікна. В залежності від стану компонента **cbAllWindows** типу **TCheckBox**, у список будуть поміщатися або заголовки всіх вікон підряд (якщо опцію встановлено), або лише видимих вікон. Перевірка того, чи вікно є видимим, здійснюється за допомогою функції **IsWindowVisible**. Крім цього, виводяться лише заголовки тих вікон, які не мають власника (тобто для яких **GetWindow(Hnd,GW_OWNER)=0**).

Заголовки знайдених вікон поміщуються у список компонента **lbWindows** типу **TListBox**.

2. Завдання на лабораторну роботу

Написати (з використанням лише функцій **Windows API**, без впровадження компонентів **VCL Delphi**) програму, яка створюватиме вікно, що міститиме інформацію про ОС чи ресурси ЕОМ згідно варіанту, приведенного у табл. №5.1.

Таблиця №5.1

Варіант	Дані, які має отримати програма
1	Тип Windows, версія, папка Windows.
2	Тип та версія ОС, назва користувача.
3	Версія та доповнення ОС, назва комп'ютера.
4	Інформація про фізичну та віртуальну пам'ять.
5	Інформація про фізичну пам'ять та параметри тома, на якому розташовано ОС.
6	Інформація про віртуальну пам'ять а також обсяг тома, на якому розташовано ОС.
7	Версія та платформа ОС, інформація про центральний процесор.
8	Папки Windows та System, список активних процесів.
9	Назви користувача та комп'ютера, список всіх вікон верхнього рівня.
10	Дані про фізичну пам'ять, список вікон верхнього рівня, які мають атрибут видимості.

Варіант	Дані, які має отримати програма
11	Дані про том, з якого запускається програма, список вікон верхнього рівня, які не мають власника.
12	Дані про центральний процесор, список процесів, запущених з папки Windows.
13	Дані про фізичну пам'ять, список процесів, запущених не з системних папок (Windows та System).
14	Інформація про обсяг місця на томі, з якого запущена програма, список процесів, у яких виконавчий модуль не є стандартним ехе-файлом (тобто має інше розширення).
15	Тип ОС, дані про мікропроцесор та фізичну пам'ять.
16	Версія та доповнення ОС, дані про віртуальну пам'ять.
17	Тип та платформа ОС, дані про мікропроцесор.
18	Всі доступні дані про том, на якому інстальовано ОС, назва користувача та комп'ютера.
19	Тип, платформа та версія ОС.
20	Список активних процесів та дані про фізичну пам'ять.
21	Тип, платформа ОС та дані про фізичну пам'ять.
22	Дані про мікропроцесор та про фізичну пам'ять.
23	Папки Windows та System, всі доступні дані про том, на якому інстальовано ОС.
24	Дані про список процесів та центральний процесор.
25	Дані про мікропроцесор та список вікон верхнього рівня.
26	Інформація про віртуальну пам'ять а також список активних процесів.
27	Версія та платформа ОС, інформація про центральний процесор.
28	Тип ОС, дані про віртуальну та фізичну пам'ять.
29	Версія та доповнення ОС, дані про мікропроцесор.
30	Тип Windows, версія та дані про том, на якому записано ОС.

3. Програмні тексти

У цьому пункті приведено текст головного модуля програми SysInfo.

```

unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, TLHelp32, Registry;

type
  TfMain = class(TForm)
    GroupBox1: TGroupBox;
    lVersionNum: TLabel;
    Label1: TLabel;
    Label2: TLabel;
    lPlatform: TLabel;
    Label3: TLabel;
    lExt: TLabel;
    Label4: TLabel;
    lCompName: TLabel;
    Label5: TLabel;
    lUser: TLabel;
    Label6: TLabel;
    lSysDir: TLabel;
    Label7: TLabel;
    lWinDir: TLabel;
    GroupBox2: TGroupBox;
    lbProcesses: TListBox;
    GroupBox3: TGroupBox;
    lbWindows: TListBox;
    cbAllWindows: TCheckBox;
    Label8: TLabel;
    lVerInfo: TLabel;
    GroupBox5: TGroupBox;
    lTotalPhys: TLabel;
    lAvailPhys: TLabel;
    lMemoryLoad: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    lTotalVirtual: TLabel;
  end;

```

```

    lAvailVirtual: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    gbHD: TGroupBox;
    Label14: TLabel;
    lVNum: TLabel;
    Label15: TLabel;
    lFileSys: TLabel;
    lTotalSpace: TLabel;
    lTotalFree: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label18: TLabel;
    lLabel: TLabel;
    GroupBox4: TGroupBox;
    Label19: TLabel;
    Label20: TLabel;
    lID: TLabel;
    lVID: TLabel;
    lMHz: TLabel;
    Label21: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure cbAllWindowsClick(Sender: TObject);
private
    { Private declarations }
public
    procedure GetMainSysInfo;
    procedure GetProcessInfo;
    procedure GetWindowsInfo;
    procedure GetMemoryInfo;
    procedure GetHDInfo;
    procedure GetCPUInfo;
end;

const
    cOsUnknown    = -1;
    cOsWin95      = 0;
    cOsWin98      = 1;
    cOsWin98SE    = 2;
    cOsWinME      = 3;
    cOsWinNT      = 4;
    cOsWin2000    = 5;
    cOsWinXP      = 6;

var
    fMain          : TfMain;
    Version         : TOSVersionInfo;
    Build          : string;
    CN              : PChar;
    CNSize          : DWORD;
    pe              : TProcessEntry32;
    hSnap           : THandle;
    SWin            : string;

implementation
{$R *.dfm}

function GetCPUSpeed: Double;
const DelayTime = 500;
var TimerHi : DWORD;
    TimerLo : DWORD;
    PriorityClass : Integer;
    Priority : Integer;
begin
    PriorityClass := GetPriorityClass(GetCurrentProcess);
    Priority := GetThreadPriority(GetCurrentThread);
    SetPriorityClass(GetCurrentProcess, REALTIME_PRIORITY_CLASS);
    SetThreadPriority(GetCurrentThread, THREAD_PRIORITY_TIME_CRITICAL);
    asm
        push 10
        call Sleep

```



```

    DW 310Fh
    MOV TimerLo, EAX
    MOV TimerHi, EDX
    push DelayTime
    call Sleep
    DW 310Fh
    SUB EAX, TimerLo
    SBB EDX, TimerHi
    MOV TimerLo, EAX
    MOV TimerHi, EDX
end;
SetThreadPriority(GetCurrentThread, Priority);
SetPriorityClass(GetCurrentProcess, PriorityClass);
Result := TimerLo / (1000.0 * DelayTime);
end;

function EnumWndFunc(Hnd:HWND;PrID:DWORD):boolean; stdcall;
var lpS : LPSTR;
begin
    Result:=true;
    GetMem(lpS,127);
    if fMain.cbAllWindows.Checked then begin
        if (GetWindowText(Hnd,lpS,127)<>0) then if lpS<>' ' then fMain.lbWindows.Items.Add(lpS);
    end else
        if (IsWindowVisible(Hnd)) and (GetWindow(Hnd,GW_OWNER)=0) and
            (GetWindowText(Hnd,lpS,127)<>0) then if lpS<>' ' then fMain.lbWindows.Items.Add(lpS);
    FreeMem(lpS,127);
end;

function GetWindowsVersion : integer;
var osVerInfo      : TOSVersionInfo;
    majorVer, minorVer : Integer;
begin
    osVerInfo.dwOSVersionInfoSize := SizeOf(TOSVersionInfo);
    if GetVersionEx(osVerInfo) then begin
        majorVer := osVerInfo.dwMajorVersion;
        minorVer := osVerInfo.dwMinorVersion;
        case osVerInfo.dwPlatformId of
            VER_PLATFORM_WIN32_NT : { Windows NT/2000/XP }
                begin
                    if majorVer <= 4 then result := cOsWinNT else
                        if (majorVer = 5) and (minorVer = 0) then result := cOsWin2000 else
                            if (majorVer = 5) and (minorVer = 1) then result := cOsWinXP else result :=
cOsUnknown;
                    end;
            VER_PLATFORM_WIN32_WINDOWS : { Windows 9x/ME }
                begin
                    if (majorVer = 4) and (minorVer = 0) then result := cOsWin95 else
                        if (majorVer = 4) and (minorVer = 10) then begin
                            if osVerInfo.szCSDVersion[1] = 'A' then result := cOsWin98SE else result := cOsWin98;
                        end else if (majorVer = 4) and (minorVer = 90) then result := cOsWinME else
                            result := cOsUnknown;
                        end;
                    else result := cOsUnknown;
                end;
            end else result := cOsUnknown;
        end;
end;

procedure TfMain.GetMainSysInfo;
var VW : integer;
begin
    with Version do begin
        dwOSVersionInfoSize:=SizeOf(TOSVersionInfo);
        GetVersionEx(Version);
        case dwPlatformId of
            VER_PLATFORM_WIN32_WINDOWS : begin lPlatform.Caption:='Win 9x';
Build:=IntToStr(LoWord(dwBuildNumber)); end;
            VER_PLATFORM_WIN32_NT : begin lPlatform.Caption:='Win NT'; Build:=IntToStr(dwBuildNumber);
end;
        end;
        lVersionNum.Caption:=IntToStr(dwMajorVersion)+'.'+IntToStr(dwMinorVersion)+'.'+Build;
        lExt.Caption:=szCSDVersion;
    end;
end;

```

```

end;

CNSize:=200;
if GetUserName(CN, CNSize) then lUser.Caption:=CN;

CNSize:=200;
if GetComputerName(CN, CNSize) then lCompName.Caption:=CN;

CNSize:=200;
if Boolean(GetSystemDirectory(CN, CNSize)) then lSysDir.Caption:=CN;

CNSize:=200;
if Boolean(GetWindowsDirectory(CN, CNSize)) then lWinDir.Caption:=CN;

VW:=GetWindowsVersion;
case VW of
  cOSUnknown : lVerInfo.Caption:='Unknown Windows';
  cOSWin95    : lVerInfo.Caption:='Windows 95';
  cOSWin98    : lVerInfo.Caption:='Windows 98';
  cOSWin98SE  : lVerInfo.Caption:='Windows 98 SE';
  cOSWinME    : lVerInfo.Caption:='Windows Millenium Edition';
  cOSWinNT    : lVerInfo.Caption:='Windows NT';
  cOSWin2000  : lVerInfo.Caption:='Windows 2000';
  cOSWinXP    : lVerInfo.Caption:='Windows XP';
end;
end;

procedure TfMain.GetProcessInfo;
begin
  pe.dwSize:=SizeOf(pe);
  hSnap:=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
  if Process32First(hSnap,pe) then
  begin
    lbProcesses.Items.Add(pe.szExeFile);
    while Process32Next(hSnap,pe) do lbProcesses.Items.Add(pe.szExeFile);
  end;
end;

procedure TfMain.GetWindowsInfo;
begin
  EnumWindows(@EnumWndFunc,0);
end;

procedure TfMain.GetMemoryInfo;
var lpBuffer : TMemoryStatus;
begin
  with lpBuffer do begin
    dwLength:=SizeOf(TMemoryStatus);
    GlobalMemoryStatus(lpBuffer);

    lTotalPhys.Caption:=IntToStr(dwTotalPhys div (1024*1024))+ ' Мбайт';
    lAvailPhys.Caption:=IntToStr(dwAvailPhys div (1024*1024))+ ' Мбайт';
    lMemoryLoad.Caption:=IntToStr(dwMemoryLoad)+ ' %';
    lTotalVirtual.Caption:=IntToStr(dwTotalVirtual div (1024*1024))+ ' Мбайт';
    lAvailVirtual.Caption:=IntToStr(dwAvailVirtual div (1024*1024))+ ' Мбайт';
  end;
end;

procedure TfMain.GetHDInfo;
var VolumeName,
  FileSystemName      : array [0..MAX_PATH-1] of Char;
  VolumeSerialNo      : DWORD;
  MaxComponentLength,
  FileSystemFlags     : DWORD;
  SC                  : pchar;
  MainDir             : string;
  FreeAvailable,
  TotalSpace          : TLargeInteger;
  TotalFree           : TLargeInteger;

```

```

begin
  MainDir:=ExtractFilePath(Application.ExeName);
  gbHD.Caption:=' Жорсткий диск '+MainDir[1]+MainDir[2]+' ';
  SC:=StrAlloc(4);
  StrPCopy(SC, MainDir[1]+MainDir[2]+MainDir[3]);
  GetVolumeInformation(SC,VolumeName,MAX_PATH,@VolumeSerialNo,
    MaxComponentLength,FileSystemFlags,FileSystemName,MAX_PATH);
  LVNum.Caption:=IntToStr(VolumeSerialNo);
  lFileSys.Caption:=FileSystemName;
  lLabel.Caption:=VolumeName;

  GetDiskFreeSpaceEx(SC, FreeAvailable, TotalSpace, @TotalFree);
  lTotalSpace.Caption:=IntToStr(TotalSpace div (1024*1024*1024))+' GB';
  lTotalFree.Caption:=IntToStr(TotalFree div (1024*1024))+' MB';
  StrDispose(SC);
end;

procedure TfMain.GetCPUInfo;
var Registry : TRegistry;
begin
  Registry:=TRegistry.Create;
  Registry.RootKey:=HKEY_LOCAL_MACHINE;
  if Registry.OpenKey('HARDWARE\DESCRIPTION\System\CentralProcessor\0\','False') then begin
    lID.Caption:=Registry.ReadString('Identifier');
    lVID.Caption:=Registry.ReadString('VendorIdentifier');
  end;
  lMHz.Caption:=IntToStr(Round(GetCPUSpeed))+' МГц';
end;

procedure TfMain.FormCreate(Sender: TObject);
begin
  CNSize:=200; GetMem(CN,CNSize);

  GetMainSysInfo;
  GetProcessInfo;
  GetWindowsInfo;
  GetMemoryInfo;
  GetHDInfo;
  GetCPUInfo;

  FreeMem(CN);
end;

procedure TfMain.cbAllWindowsClick(Sender: TObject);
begin
  lbWindows.Items.Clear;
  GetWindowsInfo;
end;

end.

```

Лабораторна робота №6

Тема: Програмування плати ЦАП/АЦП ET1270.

Мета: Вивчення принципів звертання до апаратних засобів ЕОМ. Ознайомлення з методами доступу до портів у різних видах ОС Windows.

1. Будова та характеристики плати інтерфейсу ET1270

1.1. Призначення та технічні характеристики

Інтерфейс ET1270 представляє собою плату, що вмонтовується у корпус IBM-сумісного персонального комп'ютера, і призначений для вводу/виводу аналогових сигналів в/з комп'ютера. Плата містить 8-канальний 12-бітний АЦП, 2-канальний 12-бітний ЦАП і цифровий порт вводу/виводу на 8 біт. Основні технічні дані плати ET1270 приведено у табл. №6.1.

Таблиця №6.1

Параметр	Значення
Число каналів АЦП	8
Розрядність АЦП	12 біт
Максимальна частота перетворення АЦП	75 кГц
Діапазон вхідних сигналів АЦП	0...2.5 В
Інтегральна похибка АЦП, не більше	± 1.5 біта
Диференціальна похибка АЦП, не більше	± 1 біт
Зсув нуля АЦП, не більше	± 5 біт
Вхідний струм витоку АЦП, не більше	± 10 мкА
Число каналів ЦАП	2
Розрядність ЦАП	12 біт
Діапазон установки вихідної напруги ЦАП	0...2.5 В
Типовий час установки ЦАП	15 мкс
Опір навантаження ЦАП, не більше	10 кОм
Системна шина	ISA

1.2. Роз'єми інтерфейсу

На рис. 6.1 показано зовнішній вигляд плати з розташуванням роз'ємів.

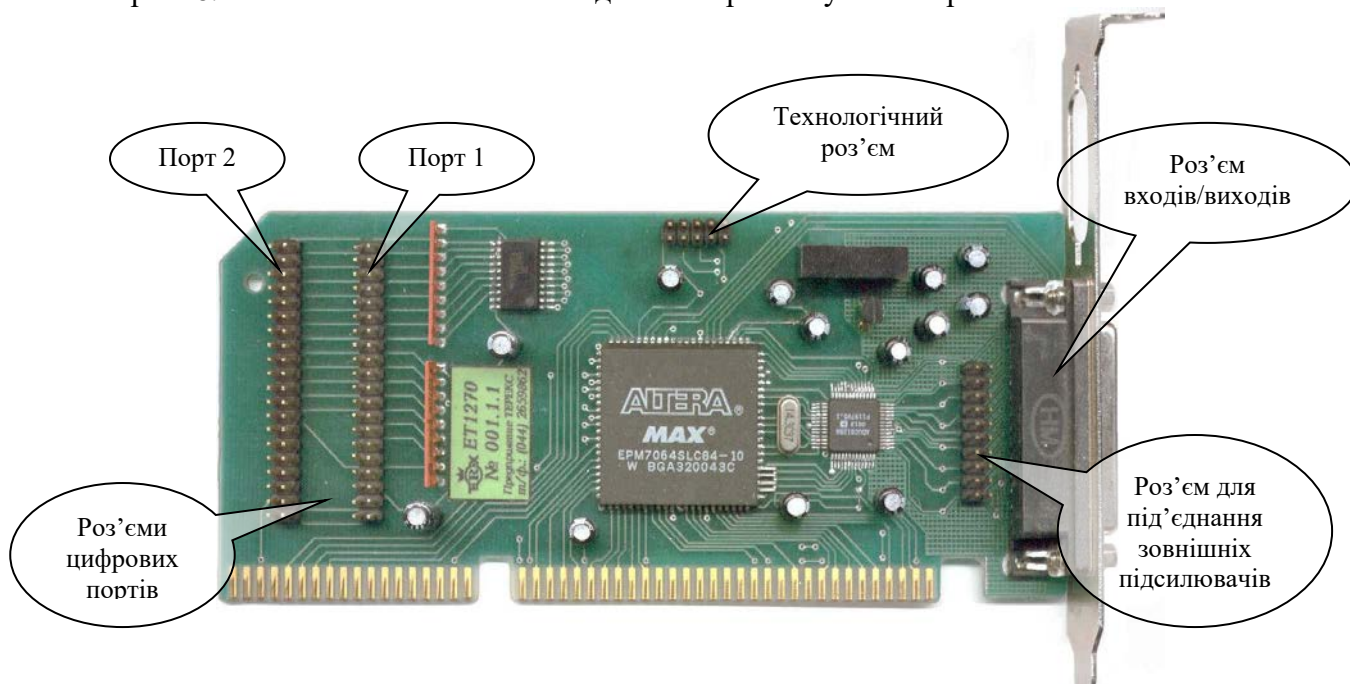


Рис. 6.1. Зовнішній вигляд плати ET1270.

Призначення контактів роз'єму входів/виходів приведено у таблиці №6.2.

Таблиця №6.2

№ контакту	Призначення	№ контакту	Призначення
1	Вхід каналу 7 АЦП	14	+15V аналогові
2	Вхід каналу 6 АЦП	15	-15V аналогові
3	Вхід каналу 5 АЦП	16	Земля аналогова
4	Вхід каналу 4 АЦП	17	Земля аналогова
5	Вихід каналу 1 ЦАП	18	Земля аналогова
6	Вихід каналу 0 ЦАП	19	Земля аналогова
7	Вхід каналу 3 АЦП	20	Земля аналогова
8	Вхід каналу 2 АЦП	21	Додатковий керуючий цифровий вихід
9	Вхід каналу 1 АЦП	22	Додатковий керуючий цифровий вихід
10	Вхід каналу 0 АЦП	23	Додатковий керуючий цифровий вихід
11	+5V шини ISA	24	Додатковий керуючий цифровий вихід
12	-12V шини ISA	25	Додатковий керуючий цифровий вихід
13	+12V шини ISA		

Призначення контактів роз'єму цифрових портів показано у таблиці №6.3.

Таблиця №6.3

№ контакту	Призначення	№ контакту	Призначення
1	D7	19	-WR
3	D6	21	ADDR0
5	D5	22	ADDR1
7	D4	23	ADDR2
9	D3	25	-PORTSEL
11	D2	27	-RESET
13	D1	36	+5V
15	D0	38	+12V
17	-RD	40	-12V
2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 24, 26, 37, 39 – DGND			

1.3. Робота з ЦАП і АЦП

Адреси регістрів плати можна представити у виді зміщення відносно базової адреси плати. Базова адреса плати після вмикання живлення дорівнює 200h, але може бути змінена програмно в діапазоні 200h...3C0h з кроком 40h. Нове значення базової адреси зберігається до вимикання живлення чи сигналу ініціалізації **RESET** шини ISA.

Робота з ЦАП і АЦП здійснюється за допомогою набору команд, перелік яких наведений у таблиці №6.4. Приведено також функції з прикладного програмного забезпечення, що реалізують відповідні команди.

Таблиця №6.4

Команда	Призначення команди	Функція
00h	Однократний запуск перетворення АЦП	ET1270_ReadADC
01h	Запуск неперервного циклічного перетворення АЦП	ET1270_StrtContADC
02h	Зупинка неперервного циклічного перетворення АЦП	ET1270_StopContADC
03h	Передати старші 4 біти даних ЦАП (канал 0)	ET1270_SetDAC
04h	Передати молодший байт даних ЦАП (канал 0)	
05h	Передати старші 4 біта даних ЦАП (канал 1)	
06h	Передати молодший байт даних ЦАП (канал 1)	
07h	Період неперервного циклічного перетворення АЦП – старший байт	ET1270_StrtContADC
08h	Період неперервного циклічного перетворення АЦП – молодший байт	

Посилання команди на ЦАП чи АЦП повинно супроводжуватися перемиканням буферних регістрів, формуванням стробуючого імпульсу готовності даних, що реалізує функція **ET1270_SetInt**.

При успішному виконанні команди встановлюється тригер готовності, стан якого повертає функція **ET1270_Ready**. Скидання тригера здійснюється читанням з порту, що відповідає базовій адресі плати.

Більш детально функції роботи з платою ET1270 (містяться у файлі **ET1270.pas**, текст якого приведено у пункті 5) описані у таблиці №6.5. Виклик функції **ET1270_SetInt** уже включений в описувані нижче функції, у зв'язку з чим вона видалена з переліку. Особливості звертання до тих чи інших функцій плати визначені апаратно при її конструюванні.

Таблиця №6.5

Назва функції (процедури)	Опис
ET1270_Init	Призначення: ініціалізує плату ET1270 Параметри: немає Значення, що повертається: якщо ініціалізація пройшла успішно – поточну базову адресу, якщо плата не відповідає – 0.
ET1270_SetBaseAddr	Призначення: встановлення базової адреси плати Параметри: значення базової адреси Значення, що повертається: немає
ET1270_ReadADC	Призначення: запуск однократного перетворення АЦП Параметри: номер каналу АЦП (0...7) Значення, що повертається: результат перетворення
ET1270_SetDAC	Призначення: установка вихідної напруги ЦАП Параметри: номер каналу (0 чи 1), код вихідної напруги (0...FFFh). Коду 0 відповідає 0 вольт вихідної напруги, FFFh – 2.5 В. Значення, що повертається: немає
ET1270_Ready	Призначення: перевірка стану тригера готовності Параметри: немає Значення, що повертається: стан тригера (false чи true)
ET1270_StrtContADC	Призначення: запуск неперервного циклічного перетворення АЦП Параметри: номер каналу (0...7), період перетворення $N=(08h...FFFFh)$. Значення періоду в мікросекундах можна обчислити за формулою: $T = N \cdot 1.68$. Після завершення чергового перетворення встановлюється тригер готовності, скинути який можна читанням результату перетворення з порту, що відповідає базовій адресі плати. Значення, що повертається: немає
ET1270_StopContADC	Призначення: зупинка неперервного циклічного перетворення АЦП Параметри: немає Значення, що повертається: немає

1.4. Робота з цифровими портами

Плата ET1270 містить 2 незалежні 8-бітні цифрові порти. На кожен порт виведено 3 лінії адреси **ADDR0...ADDR2**, що дозволяє адресувати до 8 зовнішніх пристроїв, підключених до загальної шини даних кожного порту. При вводі/виводі дані на шині кожного порту не фіксуються і стробуються сигналами відповідно читання і запису **RD** і **WR**, загальними для обох портів, а також роздільними сигналами вибору порту **PORTSEL**. Адресний простір цифрових портів наведено в таблиці №6.6 (**BaseAddr** – базова адреса плати):

Таблиця №6.6

Порт	ADDR2	ADDR1	ADDR0	Адреса
1	0	0	0	BaseAddr + 10h
	0	0	1	BaseAddr + 12h
	0	1	0	BaseAddr + 14h
	0	1	1	BaseAddr + 16h
	1	0	0	BaseAddr + 18h
	1	0	1	BaseAddr + 1Ah
	1	1	0	BaseAddr + 1Ch
	1	1	1	BaseAddr + 1Eh

Порт	ADDR2	ADDR1	ADDR0	Адреса
2	0	0	0	BaseAddr + 30h
	0	0	1	BaseAddr + 32h
	0	1	0	BaseAddr + 34h
	0	1	1	BaseAddr + 36h
	1	0	0	BaseAddr + 38h
	1	0	1	BaseAddr + 3Ah
	1	1	0	BaseAddr + 3Ch
	1	1	1	BaseAddr + 3Eh

2. Програмне звертання до портів вводу/виводу у різних видах операційної системи Windows

2.1. Організація доступу до портів

В загальному випадку особливості програмної реалізації обміну даними через порти вводу/виводу залежать від типу операційної системи, під якою буде здійснюватись передача даних.

Так, в операційних системах **Windows 95/98/ME** для передачі/прийому байта через порт можуть безпосередньо використовуватись функція **InPort** і процедура **OutPort** (тексти приведено на мові програмування **Object Pascal** з асемблерними вставками):

```
function InPort(PortNum: word): word;
begin
  try
    asm
      mov dx, PortNum
      in ax, dx
      mov @Result, ax
    end;
  except
    if ShowErrors then ShowMessage('Помилка при вводі з порта №'+IntToStr(PortNum));
  end;
end;

procedure OutPort(PortNum: word; Value: word);
begin
  try
    asm
      mov dx, PortNum
      mov ax, Value
      out dx, ax
    end;
  except
    if ShowErrors then ShowMessage('Помилка при виводі в порт №'+IntToStr(PortNum));
  end;
end;
```

Ввід з порта та вивід у порт здійснюється асемблерною частиною коду. Змінна **ShowErrors** типу **boolean** задає, чи показувати текстові повідомлення про помилки вводу/виводу.

Проте під операційними системами **Windows NT/2000/XP**, які забороняють прямо звертатись до апаратного забезпечення із-за більшої ступені захисту від несанкціонованого доступу, реалізація звертання до портів більш складна (рис.6.2).

Під операційною системою **Windows NT** звичайний користувач не має привілеїв прямого доступу до ресурсів апаратного забезпечення. Ядро **WinNT** містить карту доступу до портів вводу/виводу (**Input/Output Permissions Map- IOPM**) для кожного процесу (кожної запущеної програми). Ця карта має розмір 8Кб, і у ній кожен біт контролює доступ до одного порта із загального адресного простору 64Кб. Так, біт 0 байта 0 контролює доступ до порта 0, і т.д. Якщо при цьому біт встановлено в 1, то доступ до відповідного порта заборонено, якщо в 0 – доступ дозволено. Як правило, при запуску програми їй відмовляється в доступі до всіх портів вводу/виводу.

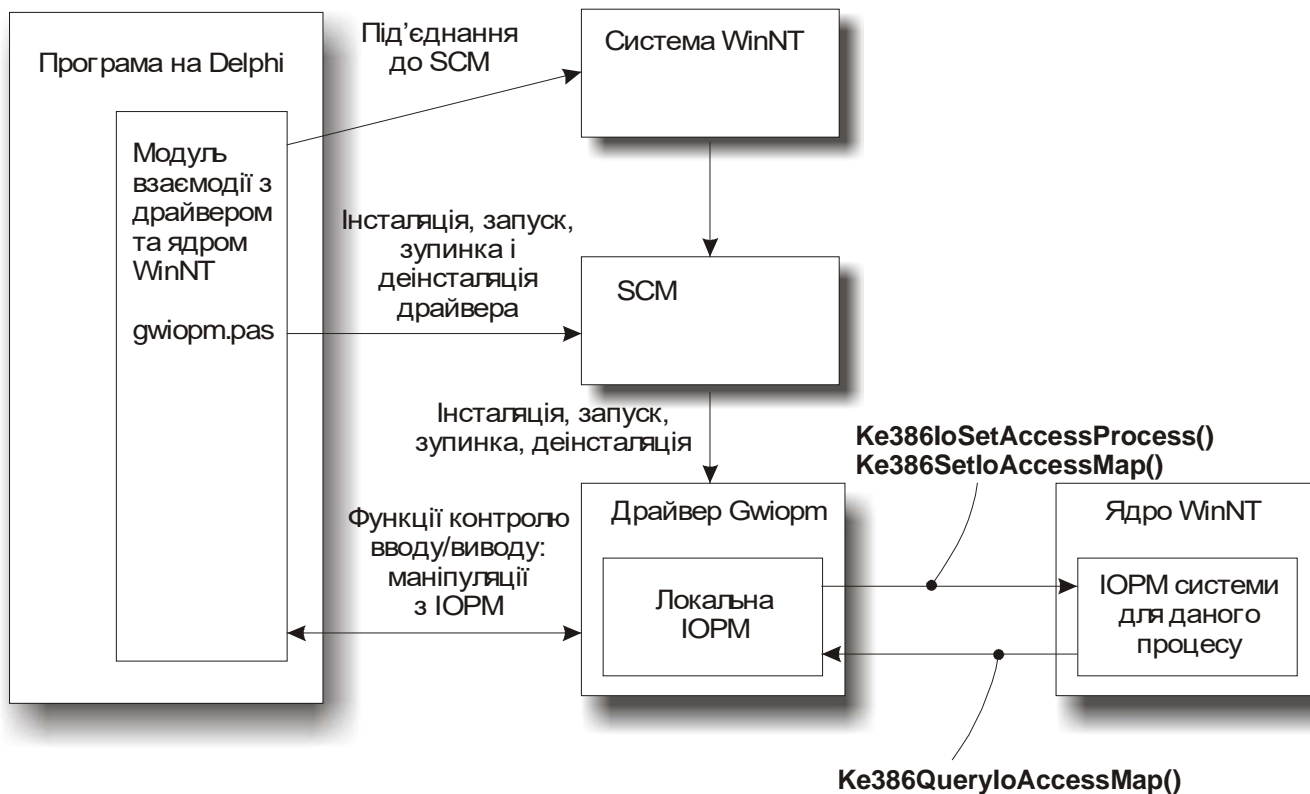


Рис.6.2. Структурна схема реалізації доступу до портів вводу/виводу для операційної системи **Windows NT**.

Для зміни **IOPM** в режимі користувача слід створити відповідний драйвер, так як драйвери мають достатні привілеїї доступу до ядра **WinNT**. Інсталяція та деінсталяція драйвера здійснюється через менеджер контролю сервісів **WinNT (Service Control Manager- SCM)**.

Таким чином, в загальному випадку організація доступу до портів вводу/виводу під операційною системою **WinNT** включає такі стадії:

1. Інсталяція та запуск драйвера, який забезпечує доступ до ядра **WinNT** та **IOPM**. Для цього використовується **SCM**.
2. Зміна за допомогою відповідних функцій драйвера таблиці **IOPM** для даного процесу таким чином, щоб забезпечити доступ до необхідних портів.
3. Ввід/вивід через порт.
4. Відновлення попереднього стану **IOPM**, зупинка та деінсталяція драйвера.

2.2. Драйвер для доступу до портів з ОС Windows NT/2000/XP

Функції ядра Windows NT, використані для драйвера, приведені у таблиці №6.7.

Таблиця №6.7

Функції ядра	Опис
<code>void Ke386IoSetAccessProcess(PEPROCESS, int);</code>	Функція надсилає до ядра інформацію про активний процес і показує йому, що ми хочемо використовувати для цього процесу спеціальну таблицю IOPM . Цілий аргумент приймає значення 1 чи 0, дозволяючи чи відключаючи спеціальну таблицю.

Функції ядра	Опис
void Ke386SetIoAccessMap(int, IOPM *);	Функція, яку драйвер використовує, щоб передати нову карту доступу (8 Кб) до ядра. Цілий аргумент означає: 1- копіює нову карту, 0- заповнює карту ядра значеннями FFh. При цьому у карті біт 1 забороняє доступ до порта, так що параметр 0 швидко забороняє доступ до всіх портів.
void Ke386QueryIoAccessMap(int, IOPM *);	Функція копіює таблицю ядра назад у локальну таблицю (при параметрі 1). Значення параметра 0 очищує таблицю драйвера значеннями FFh.

Текст драйвера реалізовано на мові **C++** і приведено у пункті 5 (файл **gwiopm.c**). У лабораторній роботі використовується скомпільований файл драйвера **gwiopm.sys**. Принципи написання драйвера у даній лабораторній роботі не розглядаються.

2.3. Робота з драйвером програми на Delphi

Модуль **gwiopm.pas** (див. пункт 5) містить об'єкт **GWIOPM_Driver**, який має кілька методів для інсталяції драйвера, підготовки таблиці доступу до портів вводу/виводу, для передачі цієї таблиці до драйвера та від нього – ядру. Призначення методів даного об'єкта приведено в таблиці №6.8.

Таблиця №6.8.

Завдання та методи	Опис
Функції під'єднання до менеджера контролю сервісів (SCM)	
OpenSCM: DWORD; CloseSCM: DWORD;	Відкриває та закриває під'єднання до SCM. Основне призначення OpenSCM - отримати дескриптор SCM, який передається у об'єкт GWIOPM_Driver для подальшого використання у функціях керування драйвером.
Функції керування драйвером	
Install(newdriverpath: string): DWORD; Start: DWORD; Stop: DWORD; Remove: DWORD;	Функції для інсталяції, запуску, зупинки і видалення драйвера. У функцію Install як параметр передається шлях до драйвера або " для значення по замовчуванню (gwiopm.sys у папці, з якої запущено програму).
Функції пристрою	
DeviceOpen: DWORD; DeviceClose: DWORD;	Спочатку треба відкрити пристрій. При цьому отримується дескриптор для подальшого використання в інших функціях. Перед видаленням драйвера треба закрити пристрій. При закриванні пристрою чи видаленні драйвера IOPM, передана ядру, не видалається.
Функції маніпулювання локальною IOPM драйвера (LIOPM)	
IOCTL_IOPMD_CLEAR_LIOPM: DWORD; IOCTL_IOPMD_SET_LIOPM(Addr: Word; B: byte): DWORD;	Очищує масив локальної карти (IOCTL_IOPMD_CLEAR_LIOPM) та встановлює заданий байт у задане значення (IOCTL_IOPMD_SET_LIOPM).

Завдання та методи	Опис
IOCTL_IOPMD_GET_LIOPMB (Addr: Word; var B: byte): DWORD; IOCTL_IOPMD_GET_LIOPMA (var A: TIOPM): DWORD;	Вибирає заданий байт (IOCTL_IOPMD_GET_LIOPMB) з локальної таблиці або цілу таблицю (IOCTL_IOPMD_GET_LIOPMA). При цьому слід пам'ятати, що дана таблиця використовується драйвером при підготовці даних, ядро ОС користується іншою таблицею.
LIOPM_Set_Ports (BeginPort: word; EndPort: word; Enable: Boolean): DWORD;	Встановлює/ скидає біт доступу для заданих портів. Функцію можна використовувати замість IOCTL_IOPMD_SET_LIOPM , якщо треба змінити лише кілька бітів.
Функції взаємодії з IOPM ядра (KIOPM)	
IOCTL_IOPMD_ACTIVATE_KIOPM : DWORD;	Передає локальну "підготовчу" таблицю драйвера ядру і робить її активною для даного процесу.
IOCTL_IOPMD_DEACTIVATE_KIOPM : DWORD;	Вказує ядру "забути" про задану процесом таблицю доступу.
IOCTL_IOPMD_QUERY_KIOPM : DWORD;	Читає таблицю ядра у драйвер. Для перегляду цієї таблиці використовується функція GET_LIOPMB .
Значення, які повертаються функціями	Функції повертають значення ERROR_SUCCESS у випадку успішного виклику, або інший код помилки, за яким функція ErrorLookup може видати текстове повідомлення про успішність проведеної операції.
ErrorLookup (ErrorNum: DWORD): string;	Функція за кодом помилки дає текстове повідомлення, яке описує успішність проведеної операції.

Створення об'єкта **GWIOPM_Driver** здійснюється при ініціалізації модуля **gwiopm**. Для цього використовується метод **Create** даного об'єкта.

3. Програма для звертання до ЦАП/АЦП плати ET1270

У пункті 5 приведено текст тестової програми для роботи з платою ЦАП/АЦП ET1270. Зовнішній вигляд головного вікна програми показано на рис. 6.3.

При запуску програми здійснюється ініціалізація плати ЦАП/АЦП та встановлення доступу до її портів. Це реалізовано у методі **FormCreate** об'єкта **TfMain**.

```

procedure TfMain.FormCreate(Sender: TObject);
begin
  // Перевіряємо версію ОС- якщо Windows NT, то пробуємо встановити дозвіл на доступ до портів
  If (GetVersion and $80000000)=0 then
  begin
    isNT:=true;
    Status := GWIOPM_Driver.OpenSCM;
    DriverStatusMessage(OP_DrvOpenSCH, Status);
    Status := GWIOPM_Driver.Install('');
    DriverStatusMessage(OP_DrvInstall, Status);
    Status := GWIOPM_Driver.Start;
    DriverStatusMessage(OP_DrvStart, Status);
    Status := GWIOPM_Driver.DeviceOpen;
    DriverStatusMessage(OP_DrvDevOpen, Status);

    SetAccessToRangeOfPorts($20);
    SetAccessToRangeOfPorts($24);
  
```

```

SetAccessToRangeOfPorts($28);
SetAccessToRangeOfPorts($2C);
SetAccessToRangeOfPorts($30);
SetAccessToRangeOfPorts($34);
SetAccessToRangeOfPorts($38);
SetAccessToRangeOfPorts($3C);
end;

BaseAddr:=ET1270_Init;

// Перевірка плати на наявність та адреси на доступність
if (BaseAddr=0)or((InPort(BaseAddr+2) and $00FF)<>$75) then
begin
  ShowMessage('Плату не знайдено або вона не відповідає. Робота буде припинена.'+
    #13'Board ET1270 not found or not respond. Program will be terminated.');
```

```

  Halt;
end;
cbBaseAddr.ItemIndex:=(BaseAddr-$200) div $40;
TimerADC.Enabled:=true;
end;

```

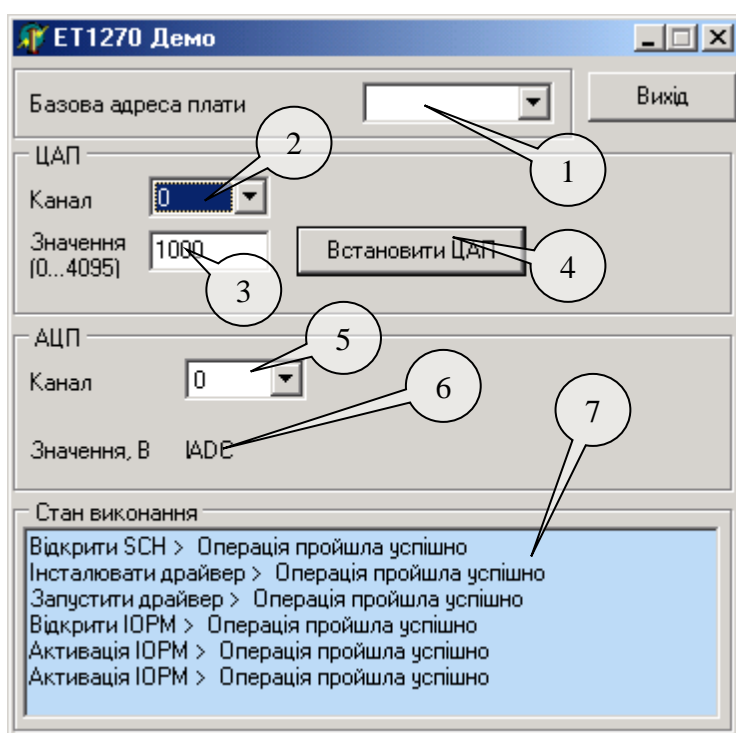


Рис. 6.3. Головне вікно програми.

Так як при роботі плати використовується певний діапазон адрес, починаючи від базової і вище, то для встановлення доступу до портів у діапазоні **NNxh** реалізовано процедуру **SetAccessToRangeOfPorts**, яка викликає метод **LIOPM_SET_PORTS** об'єкта **GWIOPM_Driver**:

```

procedure SetAccessToRangeOfPorts(Range: word);
begin
  GWIOPM_Driver.LIOPM_SET_PORTS(Range shl 4, (Range shl 4) or $F, true);
  Status:=GWIOPM_Driver.IOCTL_IOPMD_ACTIVATE_KIOPM;
  fMain.DriverStatusMessage(OP_IOPMActivate, Status);
end;

```

Наприклад, якщо у параметр **Range** цієї процедури передати значення 20h, то буде встановлено доступ до портів у діапазоні адрес від 200h до 20Fh.

Базова адреса плати (по замовчуванню 200h) може бути програмно змінена за допомогою випадального списку 1. Процедура для зміни базової адреси має наступний вид:

```

procedure TfMain.cbBaseAddrCloseUp(Sender: TObject);
var _BaseAddr : word;

```

```

begin
  _BaseAddr:=$200+cbBaseAddr.ItemIndex*$40;
  ET1270_SetBaseAddr(_BaseAddr);
  if (InPort(_BaseAddr+2) and $00FF)<>$75 then      // Перевірка адреси на доступність
  begin
    ShowMessage('Плата не відповідає. Спробуйте іншу адресу.');
```

Активний канал цифро-аналогового перетворювача вибирається за допомогою випадального списку 2, а само значення для ЦАП задається у полі 3. При цьому вхідні значення перетворювача можуть змінюватися в діапазоні від 0 до 4095 (0...FFFh). Напруга на виході ЦАП, пропорційна до введеного коду, встановлюється після натискання на кнопку 4 ("Встановити ЦАП"). При цьому коду 0 відповідає напруга 0 В, а коду FFFh – напруга 2.5 В. Для встановлення ЦАП у задане значення у програмі використовується наступний програмний код:

```

procedure TfMain.btnSetDACClick(Sender: TObject);
var Value : word;
begin
  Value:=StrToInt(e.Text);
  if Value>$FFF then ShowMessage('Very big. Not changed.');
```

Активний канал аналого-цифрового перетворювача задається у випадальному списку 5. При цьому поле 6 міститиме значення напруги (в вольтах) на вході відповідного каналу АЦП. Напрузі 0 В відповідає цифровий код 0h, а напрузі 2.5 В – код FFFh. Період оновлення інформації – 1 с. Читання значення напруги з виходу АЦП і її відображення у вікні програми здійснює метод **TimerADCTimer** об'єкта **TfMain**.

```

procedure TfMain.TimerADCTimer(Sender: TObject);
begin
  LADC.Caption:=FloatToStrF(ET1270_ReadADC(cbChannelsADC.ItemIndex)/$FFF*2.5, ffGeneral, 4, 2);
end;
```

Текстове поле 7 головного вікна програми (рис. 6.3) виконує допоміжну функцію та містить повідомлення про успішність виконання тих чи інших операцій по встановленню доступу до портів плати ЦАП/АЦП. Для виходу з програми використовується кнопка "Вихід". При знищенні головного вікна програми здійснюється деінсталяція драйвера для доступу до портів вводу/виводу:

```

procedure TfMain.FormDestroy(Sender: TObject);
begin
  // Якщо тип ОС- Windows NT, то деінстальовуємо драйвер
  if isNT then begin
    GWIOPM_Driver.DeviceClose;
    GWIOPM_Driver.Stop;
    GWIOPM_Driver.Remove;
    GWIOPM_Driver.CloseSCM;
  end;
end;
```

4. Завдання на лабораторну роботу

Написати програму (можна з використанням компонентів **VCL Delphi**), яка виконуватиме завдання згідно варіанту, приведенного у табл. №6.9. Програма повинна працювати на платформі **Win NT**. Текст драйвера розробляти не потрібно, для цього можна використати файл **gwiopm.sys**.

Таблиця №6.9

Варіант	Завдання
1, 16	Написати програму для реалізації на основі плати ET1270 генератора лінійно-змінної напруги. Частота та амплітуда імпульсів повинні регулюватися програмно.
2, 17	Написати програму для реалізації на основі плати ET1270 генератора синусоїдної напруги (лише за додатними півхвилями). Частота та амплітуда сигналу повинні регулюватися програмно.
3, 15, 18	Написати програму для реалізації на основі плати ET1270 генератора трикутної напруги. Частота, амплітуда імпульсів та співвідношення між зонами наростання і спадання імпульса повинні регулюватися програмно.
4, 19	Написати програму для реалізації на основі плати ET1270 генератора імпульсів прямокутної форми. Частота, амплітуда імпульсів, величина імпульса і проміжку між імпульсами повинні регулюватися програмно.
5, 20	Написати програму для реалізації на основі плати ET1270 генератора пилкоподібної напруги. Частота, амплітуда імпульсів та співвідношення між зонами наростання і спадання імпульса повинні регулюватися програмно.
6, 21	Написати програму для частотної модуляції сигналу, який подається на вхід АЦП.
7, 22	Написати програму для амплітудної модуляції сигналу, який подається на вхід АЦП.
8, 23	Написати програму для широтно-імпульсної модуляції сигналу, який подається на вхід АЦП.
9, 24	Написати програму для фазової модуляції сигналу, який подається на вхід АЦП.
10, 25	Написати програму для реалізації на основі плати ET1270 двоканального осцилографа. Повинно забезпечуватися масштабування за періодом та амплітудою сигналу.
11, 26	Написати програму, яка регулюватиме напругу на виході ЦАП згідно записів файлу, розміщеного на жорсткому диску. Кожен запис файлу містить два поля: величину напруги та тривалість її подання на вихід ЦАП.
12, 27, 30	Написати програму, яка регулюватиме напругу на виході АЦП згідно записів файлу, розміщеного на жорсткому диску. Кожен запис файлу містить два поля: величину цифрового коду та тривалість його подання на вихід АЦП.
13, 28	Написати програму для частотно-імпульсної модуляції сигналу, який подається на вхід АЦП.
14, 29	Написати програму для амплітудно-імпульсної модуляції сигналу, який подається на вхід АЦП.

5. Програмні тексти

У цьому пункті приведено текст програми, яка здійснює читання даних від плати ЦАП/АЦП ET1270.

Файл проекту наступний.

```
program ET1270_Demo;

uses
  Forms,
  Main in 'Main.pas' {fMain},
  ET1270 in 'ET1270.pas',
  gwiopm in 'gwiopm.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfMain, fMain);
  Application.Run;
end.
```

Модуль головного вікна програми.

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ET1270, ExtCtrls, gwiopm;
```

```

type
  TfMain = class(TForm)
    TimerADC: TTimer;
    GroupBox1: TGroupBox;
    cbChannelsDAC: TComboBox;
    Label1: TLabel;
    e: TEdit;
    Label2: TLabel;
    btnSetDAC: TButton;
    GroupBox2: TGroupBox;
    Label4: TLabel;
    GroupBox3: TGroupBox;
    Label3: TLabel;
    cbChannelsADC: TComboBox;
    Label5: TLabel;
    lADC: TLabel;
    btnExit: TButton;
    cbBaseAddr: TComboBox;
    GroupBox4: TGroupBox;
    mmStatus: TMemo;
    procedure btnSetDACClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure TimerADCTimer(Sender: TObject);
    procedure btnExitClick(Sender: TObject);
    procedure cbBaseAddrCloseUp(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure DriverStatusMessage(OP_ID: word; Status: DWORD);
  end;

var fMain : TfMain;
    isNT : boolean = false;
    Status : DWORD;

const OP_DrvOpenSCH = 1;
      OP_DrvInstall = 2;
      OP_DrvStart = 3;
      OP_DrvDevOpen = 4;
      OP_DrvDevClose = 5;
      OP_DrvStop = 6;
      OP_DrvRemove = 7;
      OP_DrvCloseSCH = 8;
      OP_IOPMActivate = 9;

type
  IntAsBytes = packed record
    b0, b1, b2, b3: byte;
  end;

implementation

{$R *.dfm}

// Установка дозволу звертання до портів діапазону адрес $NNx
procedure SetAccessToRangeOfPorts(Range: word);
begin
  GWIOPM_Driver.LIOPM_SET_PORTS(Range shl 4, (Range shl 4) or $F, true);
  Status := GWIOPM_Driver.IOCTL_IOPMD_ACTIVATE_KIOPM;
  fMain.DriverStatusMessage(OP_IOPMActivate, Status);
end;

procedure TfMain.DriverStatusMessage(OP_ID: word; Status: DWORD);
Var
  S: string;
Begin
  case OP_ID of
    OP_DrvOpenSCH : S := 'Відкрити SCH';
    OP_DrvInstall : S := 'Інсталиувати драйвер';
    OP_DrvStart : S := 'Запустити драйвер';
  end;

```



```

    OP_DrvDevOpen    : S:='Відкрити IOPM';
    OP_DrvDevClose   : S:='Закрити IOPM';
    OP_DrvStop       : S:='Зупинити драйвер';
    OP_DrvRemove     : S:='Видалити драйвер';
    OP_DrvCloseSCH   : S:='Закрити SCH';
    OP_IOPMActivate  : S:='Активация IOPM';
end;
S := S + ' > ' + GWIOPM_Driver.ErrorLookup(Status);
mmStatus.Lines.Add(S);
end;

procedure TfMain.FormCreate(Sender: TObject);
begin
    // Якщо 'Windows NT' то робимо запити на доступ до портів
    If (GetVersion and $80000000)=0 then
    begin
        isNT:=true;
        Status := GWIOPM_Driver.OpenSCM;
        DriverStatusMessage(OP_DrvOpenSCH, Status);
        Status := GWIOPM_Driver.Install('');
        DriverStatusMessage(OP_DrvInstall, Status);
        Status := GWIOPM_Driver.Start;
        DriverStatusMessage(OP_DrvStart, Status);
        Status := GWIOPM_Driver.DeviceOpen;
        DriverStatusMessage(OP_DrvDevOpen, Status);

        SetAccessToRangeOfPorts($20);
        SetAccessToRangeOfPorts($24);
        SetAccessToRangeOfPorts($28);
        SetAccessToRangeOfPorts($2C);
        SetAccessToRangeOfPorts($30);
        SetAccessToRangeOfPorts($34);
        SetAccessToRangeOfPorts($38);
        SetAccessToRangeOfPorts($3C);
    end;

    BaseAddr:=ET1270_Init;

    // Перевірка плати на наявність і адреси на доступність
    if (BaseAddr=0)or((InPort(BaseAddr+2) and $00FF)<>$75) then
    begin
        ShowMessage('Плату не знайдено або вона не відповідає. Робота буде припинена.'#13'Board
ET1270 not found or not respond. Program will be terminated.');
```

Halt;

```

    end;

    cbBaseAddr.ItemIndex:=(BaseAddr-$200) div $40;
    TimerADC.Enabled:=true;
end;

procedure TfMain.btnSetDACClick(Sender: TObject);
var Value : word;
begin
    Value:=StrToInt(e.Text);
    if Value>$FFF then ShowMessage('Very big. Not changed.');
```

ET1270_SetDAC(cbChannelsDAC.ItemIndex, Value);

```

end;

procedure TfMain.TimerADCTimer(Sender: TObject);
begin
    lADC.Caption:=FloatToStrF(ET1270_ReadADC(cbChannelsADC.ItemIndex)/$FFF*2.5, ffGeneral, 4, 2);
end;

procedure TfMain.btnExitClick(Sender: TObject);
begin
    Close;
end;

procedure TfMain.cbBaseAddrCloseUp(Sender: TObject);
var _BaseAddr : word;
begin
```

```

_BaseAddr:=$200+cbBaseAddr.ItemIndex*$40;
ET1270_SetBaseAddr(_BaseAddr);
if (InPort(_BaseAddr+2) and $00FF)<>$75 then      // Перевірка адреси на доступність
begin
  ShowMessage('Плата не відповідає. Спробуйте іншу адресу.');
```

cbBaseAddr.ItemIndex:=(BaseAddr-\$200) div \$40;

```
  Exit;
end;
BaseAddr:=_BaseAddr;
cbBaseAddr.ItemIndex:=(BaseAddr-$200) div $40;
end;

procedure TfMain.FormDestroy(Sender: TObject);
begin
  // Якщо 'Windows NT' то видаляємо драйвер
  if isNT then begin
    GWIOPM_Driver.DeviceClose;
    GWIOPM_Driver.Stop;
    GWIOPM_Driver.Remove;
    GWIOPM_Driver.CloseSCM;
  end;
end;

end.
```

Текст модуля **gwiopm.pas**, який інкапсулює засоби для роботи з драйвером вводу/виводу через порти.

```

unit gwiopm;

interface

uses Windows, Winsvc;

const IOPM_SIZE = $2000;

type
  TIOPM = array[0..IOPM_SIZE] of byte;
  PIOPM = ^TIOPM;

  TGWIOPM_Driver = class
  private
    HomeDir      : string;
    DriverDir     : string;
    DriverName    : string;
    DriverPath    : string;
    hSCMan        : SC_HANDLE; // Дескриптор Service Control Manager
    hDevice       : SC_HANDLE; // Дескриптор пристрою

    function IOCTL_IOPMD_Misc1(var RetVal: DWORD; Cmd: integer): DWORD;
    function IOCTL_IOPMD_GET_SET_LIOPM(Addr: Word; var B: byte; cmd: integer): DWORD;

  public
    LastOutBuf    : longint;
    constructor Create;

    // Взаємодія з Service Control Manager
    function OpenSCM: DWORD;
    function CloseSCM: DWORD;

    // Інсталяція/Запуск/Зупинка/Видалення драйвера
    function Install(newdriverpath: string): DWORD; { параметр по замовчуванню- '' }
    function Start:   DWORD;
    function Stop:    DWORD;
    function Remove:  DWORD;

    // Відкриття/Закриття пристрою
    function DeviceOpen: DWORD; // Отримує дескриптор hDevice
    function DeviceClose: DWORD;
```

```

// Функції IOPM
// Маніпулювання локальною IOPM (LIOPM)
function IOCTL_IOPMD_CLEAR_LIOPM: DWORD; // Встановлює локальну
                                           // таблицю для блокування
                                           // доступу до всіх портів
function IOCTL_IOPMD_SET_LIOPM(Addr: Word; B: byte): DWORD; // Встановлює байт в LIOPM
function IOCTL_IOPMD_GET_LIOPMB(Addr: Word; var B: byte): DWORD; // Отримує байт з LIOPM
function IOCTL_IOPMD_GET_LIOPMA(var A: TIOPM): DWORD; // Отримує таблицю LIOPM

function LIOPM_Set_Ports(beginPort: word; EndPort: word; Enable: Boolean): DWORD;

// Взаємодія з таблицею ядра (KIOPM)
function IOCTL_IOPMD_ACTIVATE_KIOPM: DWORD; // Копіює локальну таблицю до активної
function IOCTL_IOPMD_DEACTIVATE_KIOPM: DWORD; // Вказує ядру не використовувати таблицю
function IOCTL_IOPMD_QUERY_KIOPM: DWORD; // Читає KIOPM у LIOPM

function ErrorLookup(ErrorNum: DWORD): string;
end;

const
    DEVICE_NAME_STRING = 'gwiopm';

// Тип пристрою -- у діапазоні "Визначено користувачем"
IOPMD_TYPE = $F100;

// Коди функцій- у діапазоні від 0x800 to 0xFFFF
// Маніпулювання локальною IOPM драйвера (LIOPM)
IOCMD_IOPMD_CLEAR_LIOPM = $910;
IOCMD_IOPMD_SET_LIOPM = $911;
IOCMD_IOPMD_GET_LIOPMB = $912;
IOCMD_IOPMD_GET_LIOPMA = $913;
// Взаємодія з таблицею IOPM ядра (KIOPM)
IOCMD_IOPMD_ACTIVATE_KIOPM = $920;
IOCMD_IOPMD_DEACTIVATE_KIOPM = $921;
IOCMD_IOPMD_QUERY_KIOPM = $922;

var GWIOPM_Driver: TGWIOPM_Driver;

implementation

uses sysutils;

const
    // Взято з NTDDK
    // Типи сервісу (бітові маски)
    SERVICE_KERNEL_DRIVER = $00000001;
    SERVICE_FILE_SYSTEM_DRIVER = $00000002;
    SERVICE_ADAPTER = $00000004;
    SERVICE_RECOGNIZER_DRIVER = $00000008;

    SERVICE_DRIVER = SERVICE_KERNEL_DRIVER or
                     SERVICE_FILE_SYSTEM_DRIVER or
                     SERVICE_RECOGNIZER_DRIVER;

    SERVICE_WIN32_OWN_PROCESS = $00000010;
    SERVICE_WIN32_SHARE_PROCESS = $00000020;
    SERVICE_WIN32 = SERVICE_WIN32_OWN_PROCESS or
                    SERVICE_WIN32_SHARE_PROCESS;

    SERVICE_INTERACTIVE_PROCESS = $00000100;

    SERVICE_TYPE_ALL = SERVICE_WIN32 or
                      SERVICE_ADAPTER or
                      SERVICE_DRIVER or
                      SERVICE_INTERACTIVE_PROCESS;

    // Види запуску
    SERVICE_BOOT_START = $00000000;
    SERVICE_SYSTEM_START = $00000001;
    SERVICE_AUTO_START = $00000002;
    SERVICE_DEMAND_START = $00000003;
    SERVICE_DISABLED = $00000004;

```

```

// Типи помилок
SERVICE_ERROR_IGNORE      = $00000000;
SERVICE_ERROR_NORMAL      = $00000001;
SERVICE_ERROR_SEVERE      = $00000002;
SERVICE_ERROR_CRITICAL    = $00000003;

type
  TErrorMsg = record
    Num: integer;
    Msg: string;
  end;

const
  ErrorMsgCt = 30;
  ERROR_SCM_CANT_CONNECT = 9998;
  ERROR_NO_DEVICE_HANDLE = 9997;
  ERROR_GW_BUFFER_TOO_SMALL = 9997;
  ERROR_UNEXPECTED = 9999;

  ErrorMsgs: array[1..ErrorMsgCt] of TErrorMsg = (
    (Num: ERROR_SUCCESS           ; Msg: 'Операція пройшла успішно'),
    (Num: ERROR_INVALID_FUNCTION  ; Msg: 'Недопустима дія'),
    (Num: ERROR_ACCESS_DENIED     ; Msg: 'Access denied'),
    (Num: ERROR_CIRCULAR_DEPENDENCY ; Msg: 'Циклічна залежність'),
    (Num: ERROR_DATABASE_DOES_NOT_EXIST ; Msg: 'База не існує'),
    (Num: ERROR_DEPENDENT_SERVICES_RUNNING; Msg: 'Залежні сервіси запущені'),
    (Num: ERROR_DUP_NAME          ; Msg: 'Назва вже існує'),
    (Num: ERROR_INVALID_HANDLE    ; Msg: 'Недопустимий дескриптор'),
    (Num: ERROR_INVALID_NAME      ; Msg: 'Недопустима назва сервісу'),
    (Num: ERROR_INVALID_PARAMETER ; Msg: 'Недопустимий параметр'),
    (Num: ERROR_INVALID_SERVICE_ACCOUNT ; Msg: 'Користувач не існує'),
    (Num: ERROR_INVALID_SERVICE_CONTROL ; Msg: 'Недопустимий контрольний код сервісу'),
    (Num: ERROR_PATH_NOT_FOUND    ; Msg: 'Шлях не знайдено'),
    (Num: ERROR_SERVICE_ALREADY_RUNNING ; Msg: 'Сервіс вже запущено'),
    (Num: ERROR_SERVICE_CANNOT_ACCEPT_CTRL; Msg: 'Сервіс не може прийняти керування'),
    (Num: ERROR_SERVICE_DATABASE_LOCKED ; Msg: 'База даних захоплена'),
    (Num: ERROR_SERVICE_DEPENDENCY_DELETED; Msg: 'Посилання на неіснуючий сервіс'),
    (Num: ERROR_SERVICE_DEPENDENCY_FAIL ; Msg: 'Посилання на неробочий сервіс'),
    (Num: ERROR_SERVICE_DISABLED     ; Msg: 'Сервіс відключено'),
    (Num: ERROR_SERVICE_DOES_NOT_EXIST ; Msg: 'Сервіс не існує'),
    (Num: ERROR_SERVICE_EXISTS       ; Msg: 'Сервіс вже існує'),
    (Num: ERROR_SERVICE_LOGON_FAILED ; Msg: 'Неможливо під\'єднатися до сервісу'),
    (Num: ERROR_SERVICE_MARKED_FOR_DELETE ; Msg: 'Сервіс позначено для видалення'),
    (Num: ERROR_SERVICE_NO_THREAD    ; Msg: 'Неможливо створити потік'),
    (Num: ERROR_SERVICE_NOT_ACTIVE   ; Msg: 'Сервіс не запущено'),
    (Num: ERROR_SERVICE_REQUEST_TIMEOUT ; Msg: 'Перевищення часу очікування сервісу'),
    (Num: ERROR_GW_BUFFER_TOO_SMALL  ; Msg: 'Буфер надто малий'),
    (Num: ERROR_NO_DEVICE_HANDLE     ; Msg: 'Нема ідентифікатора пристрою'),
    (Num: ERROR_SCM_CANT_CONNECT     ; Msg: 'Неможливо під\'єднатися до Service Control Manager'),
    (Num: ERROR_UNEXPECTED           ; Msg: 'Трапилась незнайома помилка')
  );

function TGWIOPM_Driver.ErrorLookup(ErrorNum: DWORD): string;
var S: string;
    N: integer;
label foundit;
begin
  if Error <> ERROR_SUCCESS then S := 'Error: ' + IntToStr(ErrorNum) + ': ';
  for N := 1 to ErrorMsgCt do if ErrorNum = ErrorMsgs[N].Num then goto foundit;
foundit:
  if N > ErrorMsgCt then N := ErrorMsgCt;
  S := S + ErrorMsgs[N].Msg;
  result := S;
end;

// Коди IOCTL
function CTL_CODE(DeviceType: integer; func: integer; meth: integer; access: integer): DWORD;
begin
  result := (DeviceType shl 16) or (Access shl 14) or (func shl 2) or (meth);
end;

```

```

end;

const
  // Режими буферизації
  METHOD_BUFFERED      = 0;
  METHOD_IN_DIRECT      = 1;
  METHOD_OUT_DIRECT     = 2;
  METHOD_NEITHER        = 3;

  // Види доступу
  FILE_ANY_ACCESS      = 0;
  FILE_READ_ACCESS     = 1;
  FILE_WRITE_ACCESS    = 2;

constructor TGWIOPM_Driver.Create;
begin
  hSCMan := 0;
  hDevice := INVALID_HANDLE_VALUE;
  HomeDir := ExtractFilePath(ParamStr(0));
  DriverName := DEVICE_NAME_STRING;
end;

function TGWIOPM_Driver.OpenSCM: DWORD;
begin
  result := ERROR_SUCCESS;
  hSCMan := OpenSCManager(nil, nil, SC_MANAGER_ALL_ACCESS);
  if hSCMan = 0 then result := ERROR_SCM_CANT_CONNECT;
end;

function TGWIOPM_Driver.CloseSCM: DWORD;
begin
  result := ERROR_SUCCESS;
  CloseServiceHandle(hSCMan);
  hSCMan := 0;
end;

function TGWIOPM_Driver.Install(newdriverpath: string): DWORD;
var
  hService: SC_HANDLE;
  dwStatus: DWORD;
begin
  hService := 0;
  dwStatus := 0;

  if newdriverpath = '' then
  begin
    DriverDir := HomeDir;
    DriverName := DEVICE_NAME_STRING;
  end else
  begin
    DriverDir := ExtractFilePath(driverpath);
    DriverName := ExtractFileName(driverpath);
  end;
  DriverPath := DriverDir + DriverName + '.sys';

  // Додати до бази даних менеджера контролю сервісів (service control manager)
  hService := CreateService(hSCMan, PChar(DriverName), PChar(DriverName),
    SERVICE_ALL_ACCESS, SERVICE_KERNEL_DRIVER, SERVICE_DEMAND_START,
    SERVICE_ERROR_NORMAL, PChar(DriverPath),
    nil, nil, nil, nil, nil);
  if (hService = 0) then dwStatus := GetLastError() else CloseServiceHandle(hService);

  result := dwStatus;
end;

function TGWIOPM_Driver.Start: DWORD;
var
  hService: SC_HANDLE;
  dwStatus: DWORD;
  lpServiceArgVectors: PChar;

```

```

    temp: LongBool;
begin
    hService := 0;
    dwStatus := 0;
    lpServiceArgVectors := nil;

    // Отримання дескриптора сервісу
    hService := OpenService(hSCMan, PChar(DriverName), SERVICE_ALL_ACCESS);
    if hService <> 0 then
    begin
        // Запуск драйверу
        temp := StartService(hService, 0, PChar(lpServiceArgVectors));
        if not temp then dwStatus := GetLastError();
        end else dwStatus := GetLastError();

        if (hService <> 0) then CloseServiceHandle(hService);
        result := dwStatus;
    end;

function TGWIOPM_Driver.Stop:    DWORD;
var
    hService: SC_HANDLE;
    dwStatus: DWORD;
    serviceStatus: TServiceStatus;
    temp: LongBool;
begin
    hService := 0;
    dwStatus := 0;

    // Отримати дескриптор сервісу
    hService := OpenService(hSCMan, PChar(DriverName), SERVICE_ALL_ACCESS);
    if hService <> 0 then
    begin
        // Запуск драйвера
        temp := ControlService(hService, SERVICE_CONTROL_STOP, serviceStatus);
        if not temp then dwStatus := GetLastError();
        end else dwStatus := GetLastError();

        if (hService <> 0) then CloseServiceHandle(hService);
        result := dwStatus;
    end;

function TGWIOPM_Driver.Remove:    DWORD;
var
    hService: SC_HANDLE;
    dwStatus: DWORD;
    temp: LongBool;
begin
    hService := 0;
    dwStatus := 0;

    dwStatus := Stop;
    dwStatus := 0;

    // Отримати дескриптор сервісу
    hService := OpenService(hSCMan, PChar(DriverName), SERVICE_ALL_ACCESS);
    if hService <> 0 then
    begin
        temp := DeleteService(hService);
        if not temp then dwStatus := GetLastError();
        end else dwStatus := GetLastError();

        if (hService <> 0) then CloseServiceHandle(hService);
        result := dwStatus;
    end;
end;

```

```
//=====
// Функції для відкривання/закривання пристрою
//=====
function TGWIOPM_Driver.DeviceOpen: DWORD;
var dwStatus: DWORD;
begin
    dwStatus := 0;

    if hDevice <> INVALID_HANDLE_VALUE then DeviceClose;

    // Отримати дескриптор пристрою
    hDevice := CreateFile(
        '\\.\'+ DEVICE_NAME_STRING,
        GENERIC_READ or GENERIC_WRITE,
        0,
        PSECURITY_DESCRIPTOR(nil),
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        0);
        { Параметри
        { lpFileName: PChar
        { dwDesiredAccess: integer
        { dwShareMode: Integer
        { lpSecurityAttributes
        { dwCreationDisposition: DWORD
        { dwFlagsAndAttributes: DWORD
        { hTemplateFile: THandle

    if hDevice = INVALID_HANDLE_VALUE then
    begin
        dwStatus := GetLastError();
    end;

    result := dwStatus;
end;

function TGWIOPM_Driver.DeviceClose: DWORD;
var dwStatus: DWORD;
begin
    dwStatus := 0;
    if (hDevice <> INVALID_HANDLE_VALUE) then CloseHandle(hDevice);
    hDevice := INVALID_HANDLE_VALUE;
    result := dwStatus;
end;

// Функції IOPM
const
    GWIO_PARAMCOUNT = 3;
    GWIO_BYTES_IN = GWIO_PARAMCOUNT * 4;
    GWIO_BYTES_OUT = GWIO_PARAMCOUNT * 4;

type TGWIO_PARAMS = array[0..GWIO_PARAMCOUNT-1] of longint;

function TGWIOPM_Driver.IOCTL_IOPMD_Misc1(var RetVal: DWORD; Cmd: integer): DWORD;
var
    dwStatus: DWORD;
    temp: LongBool;
    BytesReturned: DWORD;
    MyControlCode: DWORD;
    InBuf: TGWIO_PARAMS;
    OutBuf: TGWIO_PARAMS;

begin
    dwStatus := 0;
    RetVal := 0;
    InBuf[0] := 0;
    InBuf[1] := 0;
    InBuf[2] := 0;

    if hDevice = INVALID_HANDLE_VALUE then
    begin
        dwStatus := ERROR_NO_DEVICE_HANDLE;
    end else
    begin
        MyControlCode := CTL_CODE(IOPMD_TYPE, Cmd, METHOD_BUFFERED, FILE_ANY_ACCESS);

        BytesReturned := 0;
        temp := DeviceIoControl(hDevice, MyControlCode,
            { Вхідний буфер (до драйвера) } @InBuf, GWIO_BYTES_IN,
```



```

        { Вихідний буфер (від драйвера) } @OutBuf, GWIO_BYTES_OUT,
        BytesReturned, nil);
    if temp then RetVal := OutBuf[0] else dwStatus := GetLastError();
end;

    result := dwStatus;
end;

function TGIOPM_Driver.IOCTL_IOPMD_CLEAR_LIOPM: DWORD;
var RetVal: DWORD;
begin
    result := IOCTL_IOPMD_Misc1(RetVal, IOCMD_IOPMD_CLEAR_LIOPM );
end;

function TGIOPM_Driver.IOCTL_IOPMD_GET_SET_LIOPM(Addr: Word; var B: byte; cmd: integer):
DWORD;
var
    dwStatus: DWORD;
    temp: LongBool;
    BytesReturned: DWORD;
    MyControlCode: DWORD;
    InBuf: TGIOW_PARAMS;
    OutBuf: TGIOW_PARAMS;

begin
    dwStatus := 0;

    if hDevice = INVALID_HANDLE_VALUE then
    begin
        dwStatus := ERROR_NO_DEVICE_HANDLE;
    end else
    begin
        MyControlCode := CTL_CODE(IOPMD_TYPE, cmd, METHOD_BUFFERED, FILE_ANY_ACCESS);

        InBuf[0] := Addr;
        InBuf[1] := B;

        BytesReturned := 0;
        temp := DeviceIoControl(hDevice, MyControlCode ,
            { Вхідний буфер (до драйвера) } @InBuf, GWIO_BYTES_IN,
            { Вихідний буфер (від драйвера) } @OutBuf, GWIO_BYTES_OUT,
            BytesReturned, nil);
        if temp then B := Lo(OutBuf[1]) else dwStatus := GetLastError();
    end;

    result := dwStatus;
end;

function TGIOPM_Driver.IOCTL_IOPMD_SET_LIOPM(Addr: Word; B: byte): DWORD;
begin
    result := IOCTL_IOPMD_GET_SET_LIOPM(Addr, B, IOCMD_IOPMD_SET_LIOPM);
end;

function TGIOPM_Driver.IOCTL_IOPMD_GET_LIOPMB(Addr: Word; var B: byte): DWORD;
begin
    result := IOCTL_IOPMD_GET_SET_LIOPM(Addr, B, IOCMD_IOPMD_GET_LIOPMB);
end;

function TGIOPM_Driver.IOCTL_IOPMD_GET_LIOPMA(var A: TIOPM): DWORD;
var
    dwStatus: DWORD;
    temp: LongBool;
    BytesReturned: DWORD;
    MyControlCode: DWORD;
    InBuf: TGIOW_PARAMS;

begin
    dwStatus := 0;

    if hDevice = INVALID_HANDLE_VALUE then
    begin

```

```

    dwStatus := ERROR_NO_DEVICE_HANDLE;
end else
begin
    MyControlCode := CTL_CODE(IOPMD_TYPE, IOCMD_IOPMD_GET_LIOPMA, METHOD_BUFFERED,
FILE_ANY_ACCESS);

    InBuf[0] := 0;
    InBuf[1] := 0;
    InBuf[2] := 0;

    BytesReturned := 0;
    temp := DeviceIoControl(hDevice, MyControlCode ,
        { Вхідний буфер (до драйвера) } @InBuf,  GWIO_BYTES_IN,
        { Вихідний буфер (від драйвера) } @A,      IOPM_SIZE,
        BytesReturned, nil);
    if not temp then dwStatus := GetLastError();
end;
    result := dwStatus;
end;

function TGIOPM_Driver.IOCTL_IOPMD_ACTIVATE_KIOPM:  DWORD;
var RetVal: DWORD;
begin
    result := IOCTL_IOPMD_Misc1(RetVal, IOCMD_IOPMD_ACTIVATE_KIOPM );
end;

function TGIOPM_Driver.IOCTL_IOPMD_DEACTIVATE_KIOPM:  DWORD;
var RetVal: DWORD;
begin
    result := IOCTL_IOPMD_Misc1(RetVal, IOCMD_IOPMD_DEACTIVATE_KIOPM );
end;

function TGIOPM_Driver.IOCTL_IOPMD_QUERY_KIOPM:  DWORD;
var RetVal: DWORD;
begin
    result := IOCTL_IOPMD_Misc1(RetVal, IOCMD_IOPMD_QUERY_KIOPM);
end;

function TGIOPM_Driver.LIOPM_Set_Ports(beginPort:  word;  EndPort:  word;  Enable:  Boolean):
DWORD;
var
    PortNum          : word;
    IOPM_Ix, IOPM_BitNum : integer;
    IOPM_Byte, Mask_Byte : byte;
    DriverResult       : DWORD;
label the_end;
begin
    DriverResult := ERROR_SUCCESS;
    IOPM_Byte := $FF;

    for PortNum := beginPort to EndPort do
    begin
        IOPM_Ix      := PortNum shr 3;
        IOPM_BitNum := PortNum and 7;  // нижні 3 біти
        Mask_Byte    := 1 shl IOPM_BitNum;

        if (PortNum = beginPort) or (IOPM_BitNum = 0) then
        begin
            DriverResult := IOCTL_IOPMD_GET_LIOPMB(IOPM_Ix, IOPM_Byte);
            if DriverResult <> ERROR_SUCCESS then goto the_end;
        end;

        if Enable then IOPM_Byte := IOPM_Byte and ($FF xor Mask_Byte)
        else IOPM_Byte := IOPM_Byte or Mask_Byte;

        if (PortNum = EndPort) or (IOPM_BitNum = 7) then
        begin
            DriverResult := IOCTL_IOPMD_SET_LIOPM(IOPM_Ix, IOPM_Byte);
            if DriverResult <> ERROR_SUCCESS then goto the_end;
        end;
    end;
end;

```

```

the_end:
    Result := DriverResult;
end;

initialization
    GWIOPM_Driver := TGWIOPM_Driver.Create;

finalization

end.

```

Текст драйвера для доступа до портів вводу/виводу. Програма написана на мові програмування C++. Для компілювання драйвера необхідна наявність Microsoft Windows NT DDK.

```

/*****
gwiopm.c  Маніпулятор доступу до портів вводу/виводу

Revision Level: See Rev Variable

98-05-23 GW Original modifications

*****/
#include <ntddk.h>

#define DEVICE_NAME_STRING    L"gwiopm"
#define IOPM_VERSION          110      // decimal
#define IOPM_TEST0             0x0123
#define IOPM_TEST1             0x1234
#define IOPM_TEST2             0x2345

// Тип пристрою          -- В діапазоні "визначені користувачем" ("User Defined")
#define IOPMD_TYPE 0xF100

// Функції IOCTL з кодами від 0x800 до 0xFFF для не-Microsoft використання.
// LIOPM означає "локальна IOPM", яка утримується даним драйвером
//-----
// Тестові функції
// повертає IOPM_TEST
#define IOCTL_IOPMD_READ_TEST          CTL_CODE(IOPMD_TYPE, 0x900, METHOD_BUFFERED, FILE_ANY_ACCESS )
// повертає IOPM_VERSION
#define IOCTL_IOPMD_READ_VERSION       CTL_CODE(IOPMD_TYPE, 0x901, METHOD_BUFFERED, FILE_ANY_ACCESS )
// Маніпулювання локальною IOPM
// блокування доступу до всіх портів В/В
#define IOCTL_IOPMD_CLEAR_LIOPM        CTL_CODE(IOPMD_TYPE, 0x910, METHOD_BUFFERED, FILE_ANY_ACCESS )
// установка байту в LIOPM
#define IOCTL_IOPMD_SET_LIOPM          CTL_CODE(IOPMD_TYPE, 0x911, METHOD_BUFFERED, FILE_ANY_ACCESS )
// взяти байт з LIOPM
#define IOCTL_IOPMD_GET_LIOPMB         CTL_CODE(IOPMD_TYPE, 0x912, METHOD_BUFFERED, FILE_ANY_ACCESS )
// прочитати поточну LIOPM в масив
#define IOCTL_IOPMD_GET_LIOPMA         CTL_CODE(IOPMD_TYPE, 0x913, METHOD_BUFFERED, FILE_ANY_ACCESS )
// Взаємодія з ядром
// копіювання LIOPM до активної таблиці
#define IOCTL_IOPMD_ACTIVATE_KIOPM     CTL_CODE(IOPMD_TYPE, 0x920, METHOD_BUFFERED, FILE_ANY_ACCESS )
// вказати NT не враховувати таблицю
#define IOCTL_IOPMD_DEACTIVATE_KIOPM   CTL_CODE(IOPMD_TYPE, 0x921, METHOD_BUFFERED, FILE_ANY_ACCESS )
// взяти системну IOPM у LIOPM
#define IOCTL_IOPMD_QUERY_KIOPM        CTL_CODE(IOPMD_TYPE, 0x922, METHOD_BUFFERED, FILE_ANY_ACCESS )

#define GWIOPM_PARAMCOUNT 3           // для більшості функцій
#define GWIOPM_PARAMCOUNT_BYTES GWIOPM_PARAMCOUNT * 4 // для більшості функцій

//-----
// масив IOPM.
// Містить 8K * 8 біт = 64K IOPM
// 0 bit -> дозволити доступ
// 1 bit -> заборонити доступ
//-----
#define IOPM_SIZE          0x2000
typedef UCHAR IOPM[IOPM_SIZE];
IOPM *IOPM_local = 0; // local version to be copied to TSS

// Функції ядра для маніпулювання таблицею доступу до портів В/В
void Ke386SetIoAccessMap(int, IOPM *);
void Ke386QueryIoAccessMap(int, IOPM *);
void Ke386IoSetAccessProcess(PEPROCESS, int);

//-----

```

```

void disp_ACTIVATE_KIOPM(void)
{
    Ke386IoSetAccessProcess(PsGetCurrentProcess(), 1);
    Ke386SetIoAccessMap(1, IOPM_local);
}
//-----
void disp_DEACTIVATE_KIOPM(void)
{
    Ke386IoSetAccessProcess(PsGetCurrentProcess(), 0);
}
//-----
void disp_QUERY_KIOPM(void)
{
    Ke386QueryIoAccessMap(1, IOPM_local);
}
//-----
void disp_CLEAR_LIOPM(void)
{
    int n;
    // Set local IOPM to all 1's
    for (n = 0; n < IOPM_SIZE; n++) {
        (*IOPM_local)[n] = 0xFF;
    }
}
//-----
NTSTATUS gwiopm_Dispatch(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP             pIrp
)
{
    int Ix, B;
    PIO_STACK_LOCATION pIrpStack;
    NTSTATUS Status;
    PULONG pIOBuffer;
    ULONG InBufferSize;
    ULONG OutBufferSize;
    ULONG OutByteCount;
    ULONG IoControlCode;

    pIrpStack = IoGetCurrentIrpStackLocation(pIrp);

    Status      = STATUS_NOT_IMPLEMENTED;
    OutByteCount = 0;

    switch (pIrpStack->MajorFunction) {
        case IRP_MJ_CREATE:
            Status = STATUS_SUCCESS;
            break;
        case IRP_MJ_CLOSE:
            Status = STATUS_SUCCESS;
            break;
        case IRP_MJ_DEVICE_CONTROL:

            InBufferSize = pIrpStack->Parameters.DeviceIoControl.InputBufferLength;

            OutBufferSize = pIrpStack->Parameters.DeviceIoControl.OutputBufferLength;

            pIOBuffer      = (PULONG)pIrp->AssociatedIrp.SystemBuffer;

            IoControlCode = pIrpStack->Parameters.DeviceIoControl.IoControlCode;
            switch (IoControlCode) {
                case IOCTL_IOPMD_READ_TEST:
                    pIOBuffer[0] = IOPM_TEST0;
                    pIOBuffer[1] = IOPM_TEST1;
                    pIOBuffer[2] = IoControlCode;
                    OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
                    Status       = STATUS_SUCCESS;
                    break;
                //-----
                case IOCTL_IOPMD_READ_VERSION:
                    pIOBuffer[0] = IOPM_VERSION;
                    pIOBuffer[2] = IoControlCode;
                    OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
                    Status       = STATUS_SUCCESS;
                    break;
                //-----
                case IOCTL_IOPMD_CLEAR_LIOPM:
                    disp_CLEAR_LIOPM();
            }
        }
    }
}

```

```

        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_SUCCESS;
        break;
    //-----
    case IOCTL_IOPMD_SET_LIOPM:
        Ix           = pIOBuffer[0] & 0x1FFF; // index, 0..0x1FFF
        B             = pIOBuffer[1] & 0xFF;   // byte, 0..0xFF
        (*IOPM_local)[Ix] = B;
        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_SUCCESS;
        break;
    //-----
    case IOCTL_IOPMD_GET_LIOPMB:
        Ix           = pIOBuffer[0] & 0x1FFF; // index, 0..0x1FFF
        B             = (*IOPM_local)[Ix];
        pIOBuffer[1] = B & 0x000000FF;        // byte.. 0..0xFF
        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_SUCCESS;
        break;
    //-----
    case IOCTL_IOPMD_GET_LIOPMA:
        if (OutBufferSize < IOPM_SIZE) {
            Status = STATUS_INVALID_PARAMETER;
        } else {
            for (Ix=0;Ix<IOPM_SIZE;Ix++) {
                ((PUCHAR)pIOBuffer)[Ix] = (*IOPM_local)[Ix];
            }
            // note, IoControlCode not echoed since buffer is filled with array
            OutByteCount = IOPM_SIZE;
            Status       = STATUS_SUCCESS;
        }
        break;
    //-----
    case IOCTL_IOPMD_ACTIVATE_KIOPM:
        disp_ACTIVATE_KIOPM();
        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_SUCCESS;
        break;
    //-----
    case IOCTL_IOPMD_DEACTIVATE_KIOPM:
        disp_DEACTIVATE_KIOPM();
        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_SUCCESS;
        break;
    //-----
    case IOCTL_IOPMD_QUERY_KIOPM:
        disp_QUERY_KIOPM();
        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_SUCCESS;
        break;
    //-----
    default:
        pIOBuffer[2] = IoControlCode;
        OutByteCount = GWIOPM_PARAMCOUNT_BYTES;
        Status       = STATUS_INVALID_DEVICE_REQUEST;
        break;
    }
    break;
}

pIrp->IoStatus.Status = Status;
pIrp->IoStatus.Information = OutByteCount;

IoCompleteRequest(pIrp, IO_NO_INCREMENT );

return Status;
}

//-----
VOID gwiopm_Unload(IN PDRIVER_OBJECT DriverObject)
{
    WCHAR DOSNameBuffer[] = L"\\DosDevices\\" DEVICE_NAME_STRING;
    UNICODE_STRING uniDOSString;

```

```

if(IOPM_local)
    MmFreeNonCachedMemory(IOPM_local, sizeof(IOPM));

RtlInitUnicodeString(&uniDOSString, DOSNameBuffer);
IoDeleteSymbolicLink (&uniDOSString);
IoDeleteDevice(DriverObject->DeviceObject);
}

//Точка входу у драйвер. Викликається раз при завантаженні драйвера
NTSTATUS DriverEntry(
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath
)
{
    PDEVICE_OBJECT deviceObject;
    NTSTATUS status;
    WCHAR NameBuffer[] = L"\\Device\\" DEVICE_NAME_STRING;
    WCHAR DOSNameBuffer[] = L"\\DosDevices\\" DEVICE_NAME_STRING;
    UNICODE_STRING uniNameString, uniDOSString;
    int n;

    IOPM_local = MmAllocateNonCachedMemory(sizeof(IOPM));
    if(IOPM_local == 0) return STATUS_INSUFFICIENT_RESOURCES;

    disp_CLEAR_LIOPM();

    RtlInitUnicodeString(&uniNameString, NameBuffer);
    RtlInitUnicodeString(&uniDOSString, DOSNameBuffer);

    status = IoCreateDevice(DriverObject, 0, &uniNameString, IOPMD_TYPE, 0, FALSE, &deviceObject);

    if(!NT_SUCCESS(status)) return status;

    status = IoCreateSymbolicLink (&uniDOSString, &uniNameString);
    if (!NT_SUCCESS(status)) return status;

    DriverObject->MajorFunction[IRP_MJ_CREATE] = gwiopm_Dispatch;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = gwiopm_Dispatch;
    DriverObject->DriverUnload = gwiopm_Unload;

    return STATUS_SUCCESS;
}

```

Лабораторна робота №7

Тема: Взаємодія із зовнішнім пристроєм через інтерфейс КОП (IEEE488) за допомогою плати ET8848.

Мета: Вивчення принципів взаємодії програми із зовнішнім пристроєм через інтерфейс КОП.

1. Структура і призначення шини КОП

Опис шини КОП³² приведено за стандартом ДСТ 26.003-80 (ГОСТ 26.003-80), який відповідає стандарту IEEE 488.

КОП повинен містити шини даних, синхронізації і управління. Опис скорочених назв ліній, зв'язок між станами ліній і їх логічними значеннями приведено в табл. 7.1.

1.1. Шина даних (ШД)

Шина даних повинна використовуватися для передачі (прийому) адресних, програмних, керуючих, основних даних і даних про стан.

Тип інформації, що передається по ШД, визначається станом лінії УП.

Період часу, протягом якого інформація на лініях даних (ЛД0 – ЛД7) дійсна, залежить від наявності сигналів на лінії СД.

1.2. Шина синхронізації (ШС)

Управління передачею інформації по лініях даних (адреси, команди, результати вимірювань або інші дані) повинно здійснюватися за допомогою трьох ліній, що входять в ШС: ГП, СД, ДП.

Лінія СД переводиться в низький стан передаючим пристроєм (“джерелом”), вказуючи на достовірність байта на ШД.

Обов'язковою умовою для переводу лінії в високий стан лінії ГП (всі приймачі прийняли і опрацювали всю попередню інформацію).

Час затримки переходу лінії СД в низький стан визначається типом збуджувачів, які використовуються у пристроях.

Лінія ГП – це лінія обміну сигналами між “приймачем” і “джерелом”, високий стан якої вказує, що “приймачі” готові до прийому інформації. Лінія ГП керує пристроями, адресованими на прийом або всіма пристроями, коли лінія УП має низький стан. Встановлення лінії в низький стан можливе лише тоді, коли лінія СД переходить в низький стан. У високий стан лінія ГП переходить після закінчення видачі сигналів на лінії ДП і після закінчення внутрішнього циклу “приймача”. Пристрої, не адресовані на прийом, повинні постійно підтримувати високий стан лінії ГП.

Наявність сигналу на лінії ДП (високий стан) вказує про кінець прийому інформації “приймачами”. Лінія ДП керує всіма пристроями, коли лінія УП має низький стан, або тими пристроями, які адресовані на прийом, коли лінія УП має високий стан. Лінія ДП приймає високий стан, коли лінії СД і ГП мають низький стан і “приймачі” здійснили прийом інформації.

Пристрої, не адресовані на прийом, повинні постійно підтримувати високий стан лінії ДП.

1.3. Шина управління (ШУ)

Шина управління повинна використовуватися для передачі керуючих сигналів між контролером і всіма іншими пристроями, з'єднаними з КОП, за допомогою ліній УП, КП, ОИ, ДУ, ЗО.

Видача сигналів на лінію УП здійснюється тільки тим пристроєм, який в даний момент виконує функцію контролера в системі. Коли на лінію УП поступає сигнал (низький стан), всі інші пристрої переходять в режим “очікування” і тільки контролер може передавати інформацію. Коли лінія УП переходить в високий стан, передають (або приймають) тільки ті пристрої, які були адресовані під час низького стану лінії УП. При цьому на “передачу” одночасно може бути ввімкнено не більше одного пристрою, в той час як на “прийом” такі обмеження не

³² КОП – канал общего пользования (рос.).

накладаються, тобто в “прийомі” одночасно може знаходитись і більше одного пристрою. Будь-який пристрій стає “джерелом”, якщо його адреса джерела розміщується на ШД в той час, коли лінія УП знаходиться в низькому стані і залишається “джерелом” до того часу, коли не будуть передані команди “не передавай”, “очистити інтерфейс” або коли по ШД передається адреса іншого “джерела”.

Таблиця №7.1

Назва шини і лінії		Позначення шини і лінії		Стан лінії ³³	Позначення стану лінії	Логічне значення стану лінії ³⁴
Українська	Міжнародна	Російське	Міжнародне			
Шина даних	Data bus	ШД				
Лінія даних 0	Data input-output 1	ЛД0	DIO1	В (Н)	$\overline{\text{ЛД}}$ (ЛД)	Л (1)
Лінія даних 1	Data input-output 2	ЛД1	DIO2	Те ж	Те ж	Те ж
Лінія даних 2	Data input-output 3	ЛД2	DIO3	--	--	--
Лінія даних 3	Data input-output 4	ЛД3	DIO4	--	--	--
Лінія даних 4	Data input-output 5	ЛД4	DIO5	--	--	--
Лінія даних 5	Data input-output 6	ЛД5	DIO6	--	--	--
Лінія даних 6	Data input-output 7	ЛД6	DIO7	--	--	--
Лінія даних 7	Data input-output 8	ЛД7	DIO8	В (Н)	$\overline{\text{ЛД}}$ (ЛД)	Л (1)
Шина синхронізації	Data byte transfer control bus	ШС				
Лінія "готов. до прийому"	Not ready for data	ГП	NRFD	В Н	ГП/ ГП	Л І
Лінія “дані прийняті”	Not data accepted	ДП	NDAC	В Н	ДП/ ДП	Л І
Лінія “супроводження даних”	Data valid	СД	DAV	В Н	СД/ СД	Л І
Шина управління	General Interface Management bus	ШУ				
Лінія “управління”	Attention	УП	ATN	В Н	УП/ УП	Л І
Лінія “кінець передачі”	End	КП	EOI	В Н	КП/ КП	Л І
Лінія “запит на обслуговування”	Service request	ЗО	SRQ	В Н	ЗО/ ЗО	Л І
Лінія “очистити інтерфейс”	Interface clear	ОИ	IFC	В Н	ОИ/ ОИ	Л І
Лінія “дистанційне управління”	Remote enable	ДУ	REN	В Н	ДУ/ ДУ	Л І

³³ Рівень напруги в високому стані (В) ≥ 2 В, в низькому (Н) стані $\leq 0,8$ В.

³⁴ І, Л – активне значення стану лінії; |І|, |Л| - пасивне значення стану лінії; І – відповідає логічній одиниці (1); Л – відповідає логічному нулю (0).

Лінія КП встановлюється “передавачем” в низький стан паралельно з передачею останнього байта даних, сигналізуючи, що даних більше немає. Вона може встановлюватись в низький стан також пристроєм управління при реалізації ним паралельного опитування (в цьому випадку КП інтерпретується як “ідентифікація” (ІДТ).

Коли лінія ОИ, яка використовується при запуску системи, переходить в низький стан, припиняється вся діяльність каналу передачі інформації, всі пристрої звільняють себе від адрес і переходять у стан холостого ходу.

При встановленні лінії ДУ в низький стан пристрій отримує дозвіл на перемикання управління з “місцевого” на “дистанційне”. При високому стані лінії ДУ пристрій повинен знаходитись в “місцевому” управлінні.

Лінія ЗО переходить в низький стан в тому випадку, коли будь-який пристрій посилає контролеру сигнал запиту на обслуговування.

1.4. Інтерфейсні повідомлення

Для керування функціями інтерфейсу використовуються інтерфейсні повідомлення (команди). Ці повідомлення передаються при низькому стані лінії УП та у пристрій не проходять. Кодування інтерфейсних повідомлень приведено у таблиці №7.2.

Таблиця №7.2 ³⁵

Назва багатолінійних команд		Позначення багатолінійних команд		Логічне значення на лініях даних							
Українська	Міжнародна	Російське	Міжнародне	ЛД7	ЛД6	ЛД5	ЛД4	ЛД3	ЛД2	ЛД1	ЛД0
Група адресних команд	Adressed command group	ГАК	ACG	X	0	C	0	X	X	X	X
Група універсальних команд	Universal command group	ГУК	UCG	X	0	0	1	X	X	X	X
Група адресів приймачів	Listen address group	ГАП	LAG	X	0	1	X	X	X	X	X
Група адресів джерел	Talk address group	ГАИ	TAG	X	1	0	X	X	X	X	X
Група вторинних команд або адресів	Secondary command group	ГБК	SCG	X	1	1	X	X	X	X	X
Перехід на місцеве управління	Go to local	ПНМ	GTL	X	0	0	0	0	0	0	1
Скид адресний	Selected device clear	СБА	SDC	X	0	0	0	0	1	0	0
Конфігурація паралельного опитування	Parallel poll configure	КПР	PPC	X	0	0	0	0	1	0	1
Запуск пристрою	Group execute trigger	ЗАП	GET	X	0	0	0	1	0	0	0
Взяти управління	Take control	ВУП	TCT	X	0	0	0	1	0	0	1
Скид універсальний	Device clear	СБУ	DCL	X	0	0	1	0	1	0	0

³⁵ Позначення: X – лінію використовувати не обов’язково; C – біт “зчитування”, що приписує істинне значення біту стану при паралельному опитуванні. Паралельне опитування можливе, якщо цей біт співпадає з бітом, що видається пристроєм; П – біти, що приписують лінію даних, по якій пристрій повинен видавати місцеве повідомлення реакції на паралельне опитування; Н – біти повідомлення, на які приймач не повинен реагувати.

Назва багатолінійних команд		Позначення багатолінійних команд		Логічне значення на лініях даних							
Українська	Міжнародна	Російське	Міжнародне	ЛД7	ЛД6	ЛД5	ЛД4	ЛД3	ЛД2	ЛД1	ЛД0
Деконфігурація паралельного опитування	Parallel poll unconfigure	ДПР	PPU	X	0	0	1	0	1	0	1
Відпирання послідовного опитування	Serial poll enable	ОПО	SPE	X	0	0	1	1	0	0	0
Запирання послідовного опитування	Serial poll disable	ЗПО	SPD	X	0	0	1	1	0	0	1
Запирання місцевого управління	Local lockout	ЗПМ	LLO	X	0	0	1	0	0	0	1
Запирання паралельного опитування	Parallel poll disable	ЗПР	PPD	X	1	1	1	Н	Н	Н	Н
Відпирання паралельного опитування	Parallel poll enable	ОПР	PPE	X	1	1	0	С	П	П	П
Не приймати	Unlisten	НПМ	UNL	X	0	1	1	1	1	1	1
Не передавати	–	НПД	–	X	1	0	1	1	1	1	1

1.5. Часова послідовність процесу синхронізації

На рис. 7.1 показана часова послідовність сигналів в шині синхронізації для одного джерела і багатьох приймачів сигналів.

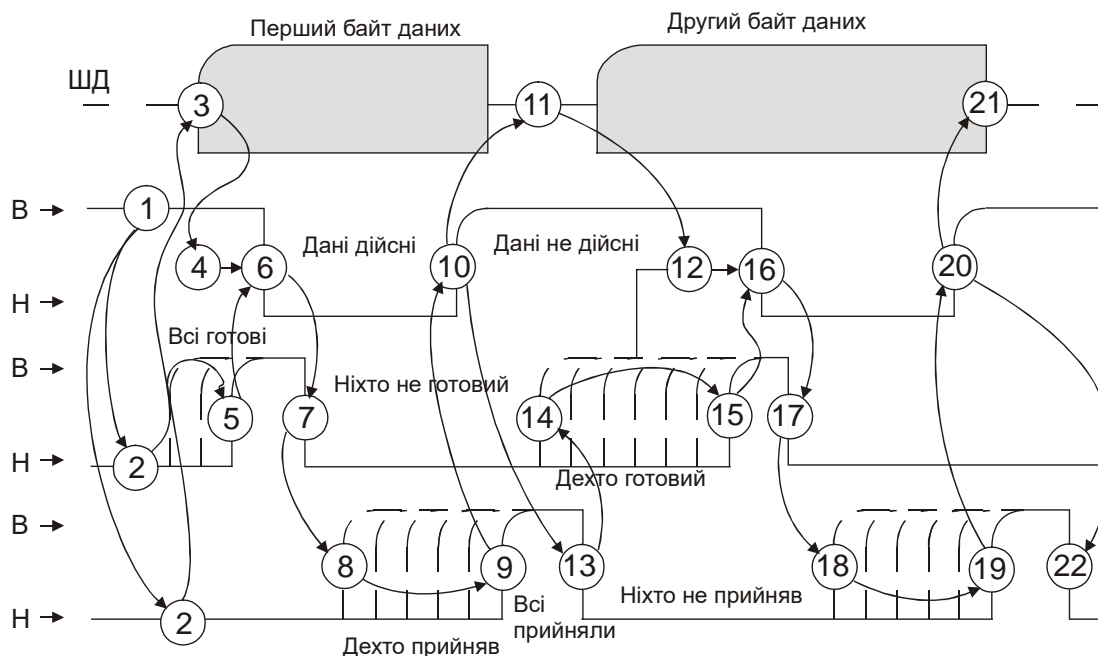


Рис.7.1. Часова послідовність процесу синхронізації

- 1 – джерело встановлює СД у високий стан (дані дійсні);
- 2 – “приймачі” встановлюють ДП і ГД у низький стан (нічого не прийнято, ніхто не готовий);
- 3 – “джерело” перевіряє помилку (ДП і ГП – в високому стані), потім посилає байт даних на ШД;

4 – “джерело” затримує підтвердження істинності даних для того, щоб дані могли поступити через ШД на всі “приймачі” (щоб дані встановились);
 5 – всі “приймачі” вказали на готовність прийому першого байту даних: ГП переходить в високий стан;
 6 – після прийому ГП “джерело” встановлює СД в низький стан для індикації того, що дані на ШД встановлені і істинні;
 7 – після переходу СД в низький стан “приймач” переводить ГП в низький стан (не готовий до прийому), тоді приймає дані. Всі інші приймачі працюють аналогічним чином зі своєю швидкодією;
 8 – перший “приймач” встановлює ДП в високий стан для вказання того, що він прийняв дані, але лінія ДП залишається в низькому стані, так як інші “приймачі” утримують її в цьому стані;
 9 – останній “приймач” встановлює ДП в високий стан, вказуючи, що він і всі інші прийняли дані. Лінія ДП при цьому переходить в високий стан;
 10 – “джерело”, прийнявши інформацію про те, що ДП знаходиться в високому стані, встановлює СД в високий стан. Це показує “приймачам”, що дані на ШД повинні розглядатися як недійсні;
 11 – “джерело” змінює дані на ШД;
 12 – “джерело” затримує підтвердження істинності даних для того, щоб дані встановились на лініях ШД;
 13 – перший “приймач”, прийнявши інформацію про те, що СД знаходиться в високому стані, встановлює ДП в низький стан для підготовки до наступного циклу;
 14 – перший “приймач” показує, що він готовий до прийому наступного байту даних, встановлюючи ГП в високий рівень, але лінія ГП залишається в низькому стані, оскільки інші “приймачі” утримують її в цьому стані;
 15 – останній “приймач” показує, що він і всі інші готові до прийому наступного байту даних, встановлюючи ГП в високий стан;
 16 – “джерело”, прийнявши інформацію про те, що ГП знаходиться в високому стані, встановлює СД в низький стан, вказуючи цим, що дані на ШД встановлені і істинні;
 17 – перший “приймач” встановлює ГП в низький стан, після чого приймає дані;
 18, 19, 20 – відповідають позиціям 8, 9, 10;
 21 – “джерело” знімає байт даних (кінець передачі) з ШД після встановлення СД у високий стан;
 22 – “приймач”, прийнявши інформацію про те, що СД знаходиться у високому стані, встановлює ДП в низький стан для підготовки до наступного циклу;

1.6. Часова послідовність ідентифікації при запиті на обслуговування

Послідовність сигналів для такого випадку показана на рис.7.2.

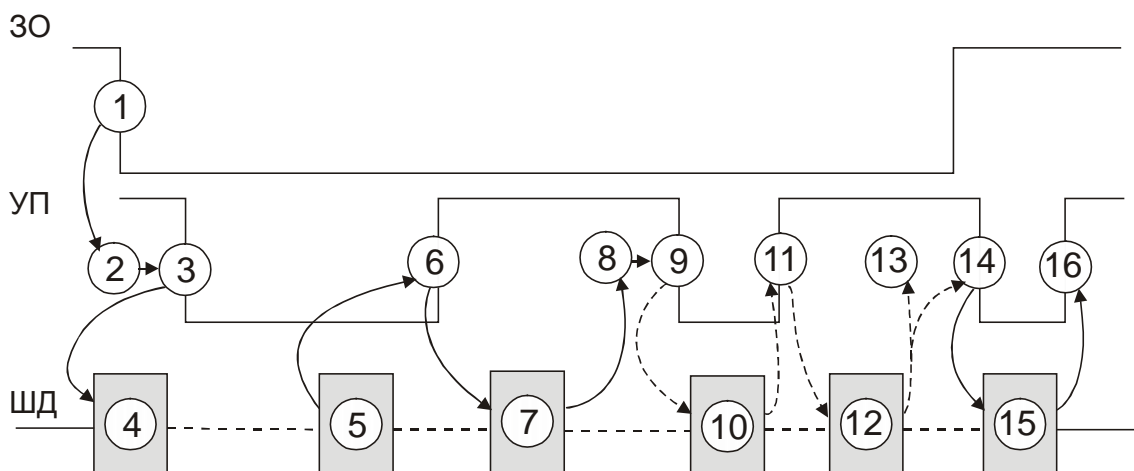


Рис.7.2. Часова послідовність процесу ідентифікації при запиті на обслуговування

- 1 – прилад робить „запит на обслуговування” шляхом встановлення лінії ЗО в робочий стан;
- 2 – невизначений проміжок часу (залежить від програми), поки контролер не ввімкне цикл ідентифікації ЗО;
- 3 – контролер встановлює УП в низький стан, щоб передати необхідні команди;
- 4 – контролер посилає універсальну команду „відпирання послідовного опитування”;
- 5 – контролер посилає адрес на передачу потенціальному запитувачу;
- 6 – контролер встановлює УП в високий стан для того, щоб адресоване „джерело” могло відіслати свій байт стану;
- 7 – адресоване „джерело” посилає один байт даних про стан;
- 8 – контролер перевіряє байт даних про стан і інтерпретує його таким чином: біт 6=0 – не робить запиту на обслуговування, біт 6=1 – робить запит на обслуговування, біт з 5-го по 0-ий – дані про стан;
- 9 – контролер приймає рішення: якщо всі необхідні прилади опитані, переходить до виконання позиції 14; якщо необхідно опитати інші прилади – переходить до виконання позиції 10;
- 10 – контролер посилає адресу на передачу другому потенціальному запитувачу (як в позиції 5);
- 11 – контролер встановлює УП в високий стан для того, щоб адресоване „джерело” могло послати свій байт стану (як в позиції 6);
- 12 – аналогічно позиції 7;
- 13 – контролер перевіряє дані про стан (як в позиції 8), а потім повертається до виконання позиції 9;
- 14 – всі прилади, які цікавлять контролер, опитані. Контролер встановлює УП в низький стан для того, щоб команда або адреса передавались до нього по ШД;
- 15 – контролер посилає універсальну команду для закінчення послідовного опитування;
- 16 – контролер встановлює УП у високий стан, знову починається процес ідентифікації.

2. Опис вимірювача імітансу Е7-14

2.1. Призначення та основні технічні характеристики

Вимірювач імітансу **Е7-14** призначений для вимірювання імітансних параметрів електрорадіокомпонентів: резисторів, конденсаторів, катушок індуктивності. Головні області використання пристрою наступні: вимірювання імітансних параметрів радіоелементів в лабораторних умовах, на вхідному та виробничому контролі радіоелементів. Прилади можуть працювати в системах, організованих в лінію колективного користування.

Робочі частоти пристрою – 0.1; 1; та 10 кГц з похибкою встановлення не більше 0.01%. Прилади мають два рівні вимірювального сигналу: (2 ± 0.4) V (високий рівень) та (40 ± 8) mV середньоквадратичного значення (низький рівень).

Прилад вимірює наступні імітансні параметри:

Назва параметра	Позначення
Паралельну і послідовну індуктивність	LP, LS
Паралельну і послідовну ємність	CP, CS
Паралельний і послідовний опір	RP, RS
Паралельну провідність	G
Фактор втрат	D
Добротність	Q

Прилади **Е7-14** забезпечують програмування у відповідності з таблицею 7.3. При цьому набір командних кодів передається у пристрій у виді команд інтерфейсу **КОП**.

Таблиця №7.3

Програмована функція	Програмний код
Параметр А:	
L	P0
C	P1
R	P2
Параметр В:	
D	P3
Q	P4
R/G	P6
L/C	P5
Еквівалентна схема:	
Автоматичний вибір	C0
Послідовна	C1
Паралельна	C2
Межа вимірювання:	
1	R1
2	R2
3	R3
4	R4
5	R5
6	R6
7	R7
8	R8
Автоматичний вибір межі	R0

Програмована функція	Програмний код
Усереднення:	
Викл	S0
10	S1
100	S2
Частота:	
100 Hz	F0
1 kHz	F1
10 kHz	F2
Рівень сигналу	
40 mV	L0
2 V	L1
Включений	T1
Провести однократне вимірювання	EX
Провести вимірювання початкових параметрів (K3)	ZS
Провести вимірювання початкових параметрів (XX)	ZO
Видати інформацію для навчання	KY
Встановити вихідний стан	HM
Ввід	IN
Встановити буфер даних на початок	AG
Заборонити видачу ЗО за всіма причинами, крім аварії	Q0
Дозволити видачу ЗО	Q1

Інформація про вимірювані параметри А або В видається в канал загального користування (КОП) у такому форматі:

X XX NNN.NNNENN ПС
1 2 3 4

де 1 – символ N для нормального вимірювання або символ F для неправильного вимірювання;

2 – заголовок (назва інформації) у відповідності з таблицею №7.4;

3 – число в експоненціальному представленні; положення десяткової точки повинно відповідати її положенню на індикаторі результату вимірювань;

4 – символ закінчення даних, повинен передаватися одночасно з видачею сигналу КП на відповідну лінію КОП.

Таблиця №7.4

Назва інформації	Заголовок
Індуктивність послідовна	LS
Ємність послідовна	CS
Опір послідовний	RS
Індуктивність паралельна	LP
Ємність паралельна	CP

Назва інформації	Заголовок
Опір паралельний	RP
Провідність паралельна	GP
Тангенс кута втрат	DD
Добротність	QQ

Пристрій забезпечує видачу в КОП сигналу **ЗАПИТ ОБСЛУГОВУВАННЯ (ЗО)** при таких причинах:

- прилад несправний;
- нормальне завершення вимірювання, дані готові;
- невірне програмування приладу;
- перезавантаження приладу.

2.2. Особливості роботи пристрою з КОП

Коди програмування приведені в таблиці №7.3.

Пристрій здійснює прийом програмних кодів в спеціально відведену область пам'яті (буфер вхідних даних) обмеженого обсягу. Для уникнення переповнення буфера необхідно в кінці рядка кодів ставити символ вводу IN. Виявивши ці символи в командному рядку, внутрішня програма, що обслуговує КОП, виконує встановлення запрограмованих параметрів і встановлює вказівник буфера на початок.

Якщо в рядку кодів зустрілись символи EX (це означає запуск вимірювання), то виконання буде проведено після IN.

Приклад. Запрограмувати частоту 1 kHz:

F1	код частоти розміщений в буфері, на виході генератора сигнала залишається попередня частота;
F1 IN	код частоти розміщений в буфері, генератор перепрограмований на частоту 1 kHz, буфер очищений.

Для економії часу системного контролера бажано рідше використовувати в програмі символи IN. Від моменту видачі IN і до появи готовності програмувати прилад забороняється.

Встановлення параметру А (В) для вимірювання виконується за останнім кодом, що зустрівся в стрічці кодів, з групи параметрів А (В), наприклад, R0P5P1P6EXIN будуть виміряні параметри С (P1) і R/G (P6).

Програмування останнього параметру А або В визначає видачу в КОП і на дисплей приладу, наприклад, P1P6EXIN – вимірюються параметри С і R/G. В КОП буде видаватися параметр R/G (P6), а для отримання через КОП параметра С (P1) потрібно задати P1 IN без запуску приладу.

Без запуску приладу можна отримати через КОП тільки ті параметри А і В, які згадувалися останніми в рядку кодів разом з EX.

Використання лінії ЗО і послідовного опитування позбавляє системного програміста від необхідності слідкувати за часом вимірювання і гарантує передачу приладом істинних даних.

Сигнал ЗО, виданий приладом в лінію КОП, знімається в таких випадках:

1. При передачі команд СБА (Скидання адресне), СБУ (Скидання універсальне) або стрічки HMIN.
2. При адресації на передачу в циклі послідовного опитування при будь-якій причині, що викликала ЗО.
3. (ЗО – ДАНІ ГОТОВІ) при передачі останнього байту формату нових даних або при черговому запуску.
4. (ЗО – ПЕРЕЗАВАНТАЖЕННЯ) при черговому запуску (EX або ЗАП.У (Запуск пристрою)).
5. (ЗО – НЕСПРАВНІСТЬ) – тільки у випадках 1 або 2.

Індикатор ЗО на передній панелі приладу не відображає стан лінії ЗО, а вказує лише на те, що причина, яка викликала запит обслуговування, все ще присутня.

Якщо, наприклад, було лише ЗО – ДАНІ ГОТОВІ, то сигнал в лінії знімається при послідовному опитуванні, а індикатор ЗО погасне лише після передачі даних або при видачі нової команди ЗАП.У або при видачі ОИ.

Використання стрічки кодів AGIN (встановити вказівник буфера даних на початок) дозволяє отримати формат даних повторно, якщо було асинхронне переривання передачі. Для цього прилад адресується на прийом, видаються коди AGIN, прилад адресується на передачу і видає формат даних.

Використання стрічки кодів KYIN дозволяє приладу видавати стан всіх програмованих через КОП режимів приладу, попередньо набраних з передньої панелі або пульта управління. Для цього прилад адресується на прийом, видається стрічка кодів KYIN, видається сигнал ЗО (ДАНІ ГОТОВІ), прилад адресується на передачу і видає наступний формат даних:

де X – перший натиснутий параметр P , Y – другий натиснутий параметр P (цифра).

3. Будова та характеристики плати інтерфейсу ET8848

Інтерфейс **ET8848** представляє собою плату, що вмонтовується в корпус IBM-сумісного персонального комп'ютера і призначена для керування і зняття даних з вимірювальних пристроїв з байт-послідовним і біт-паралельним обміном інформацією. Технічні характеристики інтерфейсу відповідають вимогам до сумісності до пристроїв, які мають інтерфейс **КОП (ДСТ 26.003-80)**.

3.1. Основні технічні дані і характеристики

Розрядність шини даних, ліній – 8;

Розрядність шини синхронізації, ліній – 3;

Розрядність шини керування, ліній – 5;

Тип збуджувачів шини даних :

- 'із третім станом ';

Тип збуджувачів шини синхронізації :

- 'із третім станом ';
- 'з відкритим колектором';

Тип збуджувачів шини керування :

- 'із третім станом ';
- 'з відкритим колектором';

Навантажувальна здатність збуджувачів обох типів, не менше – 40 ма.

3.2. Опис плати інтерфейсу

На рис. 7.3. представлено зовнішній вигляд плати . Базова адреса інтерфейсу задається за допомогою перемичок J1-J4, розташованих на платі.

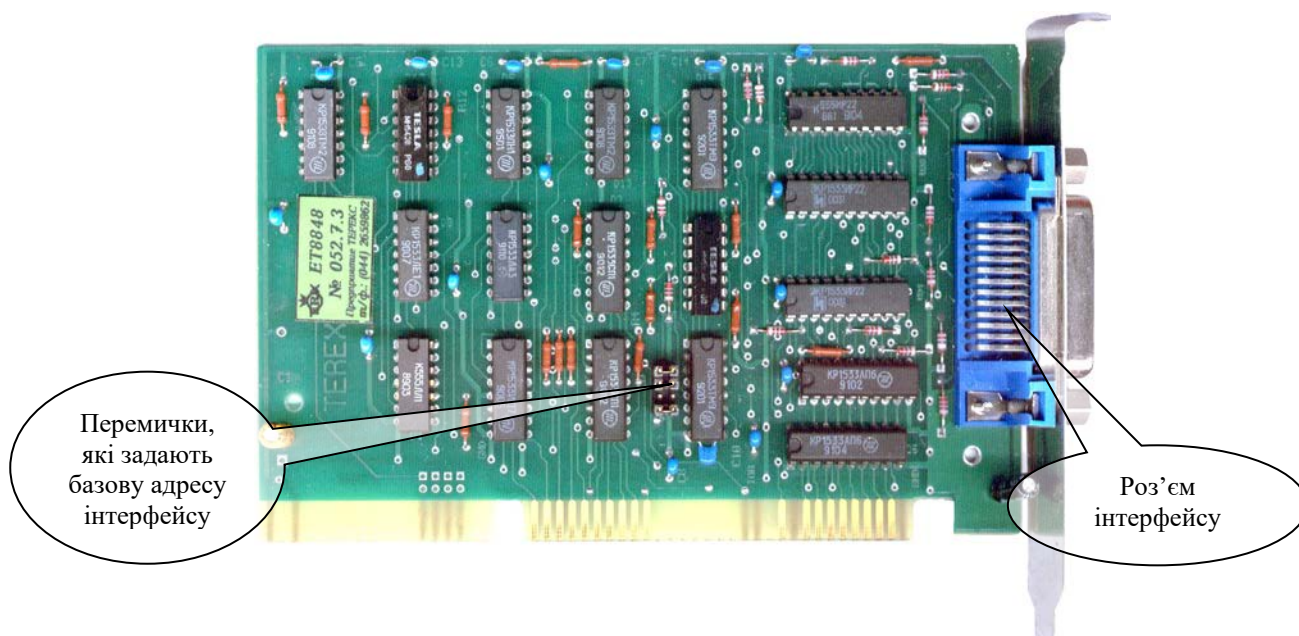


Рис. 7.3. Зовнішній вигляд плати **ET8848**.

Відповідність між конфігурацією встановлених перемичок на платі і його базовою адресою приведено в таблиці 7.5.

Таблиця №7.5

J4	J3	J2	J1	Базова адреса
ON	ON	ON	ON	\$100
OFF	ON	ON	ON	\$110
ON	OFF	ON	ON	\$120
OFF	OFF	ON	ON	\$130
ON	ON	OFF	ON	\$140
OFF	ON	OFF	ON	\$150
ON	OFF	OFF	ON	\$160
OFF	OFF	OFF	ON	\$170

J4	J3	J2	J1	Базова адреса
ON	ON	ON	OFF	\$180
OFF	ON	ON	OFF	\$190
ON	OFF	ON	OFF	\$1A0
OFF	OFF	ON	OFF	\$1B0
ON	ON	OFF	OFF	\$1C0
OFF	ON	OFF	OFF	\$1D0
ON	OFF	OFF	OFF	\$1E0
OFF	OFF	OFF	OFF	\$1F0

Розпаювання контактів на роз'ємі інтерфейсу приведено в таблиці №7.6.

Таблиця №7.6.

Номер контакту	Лінія сигналів
1	ЛД0
2	ЛД4
3	ЛД1
4	ЛД5
5	ЛД2
6	ЛД6
7	ЛД3
8	ЛД7
9	КП
10	ДУ
11	СД
12	ЗАГАЛЬНИЙ

Номер контакту	Лінія сигналів
13	ГП
14	ЗАГАЛЬНИЙ
15	ДП
16	ЗАГАЛЬНИЙ
17	ОИ
18	ЗАГАЛЬНИЙ
19	ЗО
20	ЗАГАЛЬНИЙ
21	УП
22	ЗАГАЛЬНИЙ
23	ЗАГАЛЬНИЙ
24	ЗАГАЛЬНИЙ

Структурна схема інтерфейсу **ET8848** приведена на рис. 7.4. Призначення сигналів, які формуються різними блоками пристрою, описані в п.1.

3.3. Адресний простір інтерфейсу

Значення адрес представлені у виді зсуву відносно базової адреси плати.

Поняття високих і низьких рівнів сигналів відповідає рівням ТТЛ. Адреси доступних для читання/ запису портів та призначення їх інформаційних бітів приведено в таблиці №7.7.

Таблиця №7.7

Адреса та призначення порта	Призначення інформаційних бітів порта
ADR = \$ 0 , читання/запис. Порт шини даних КОПа.	<div style="text-align: center;"> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 2px 10px;">ст арший біт</div> <div style="border: 1px solid black; padding: 2px 10px;">молодший біт</div> </div> <div style="display: flex; justify-content: center; align-items: center; margin-top: 10px;"> <div style="border: 1px solid black; padding: 2px 10px;">D7</div> <div style="border: 1px solid black; padding: 2px 10px;">D6</div> <div style="border: 1px solid black; padding: 2px 10px;">D5</div> <div style="border: 1px solid black; padding: 2px 10px;">D4</div> <div style="border: 1px solid black; padding: 2px 10px;">D3</div> <div style="border: 1px solid black; padding: 2px 10px;">D2</div> <div style="border: 1px solid black; padding: 2px 10px;">D1</div> <div style="border: 1px solid black; padding: 2px 10px;">D0</div> </div> <p style="text-align: center; margin-top: 5px;">D7 - D0 - біти шини даних КОПа</p> </div>

Адреса та призначення порта	Призначення інформаційних бітів порта
ADR = \$ 1 , читання. Порт шини керування/ синхронізації КОПа.	<div style="text-align: center;"> <div style="display: flex; justify-content: space-around; margin-bottom: 10px;"> <div style="border: 1px solid black; padding: 2px 10px;">старший біт</div> <div style="border: 1px solid black; padding: 2px 10px;">молодший біт</div> </div> <div style="display: flex; justify-content: space-around; border: 1px solid black; padding: 5px;"> <div style="border: 1px solid black; padding: 2px 10px;">УП</div> <div style="border: 1px solid black; padding: 2px 10px;">ЗО</div> <div style="border: 1px solid black; padding: 2px 10px;">ОИ</div> <div style="border: 1px solid black; padding: 2px 10px;">КП</div> <div style="border: 1px solid black; padding: 2px 10px;">ДП</div> <div style="border: 1px solid black; padding: 2px 10px;">ГП</div> <div style="border: 1px solid black; padding: 2px 10px;">ОД</div> <div style="border: 1px solid black; padding: 2px 10px;">ДЕ</div> </div> </div> <p>ДЕ - біт тригера ознаки строба даних. Встановлюється в '1' низьким рівнем на вході СД, у '0' – командою читання з даного порту.</p> <p>СД...УП - біти, які приймають значення відповідних сигналів на роз'ємі інтерфейсу.</p>

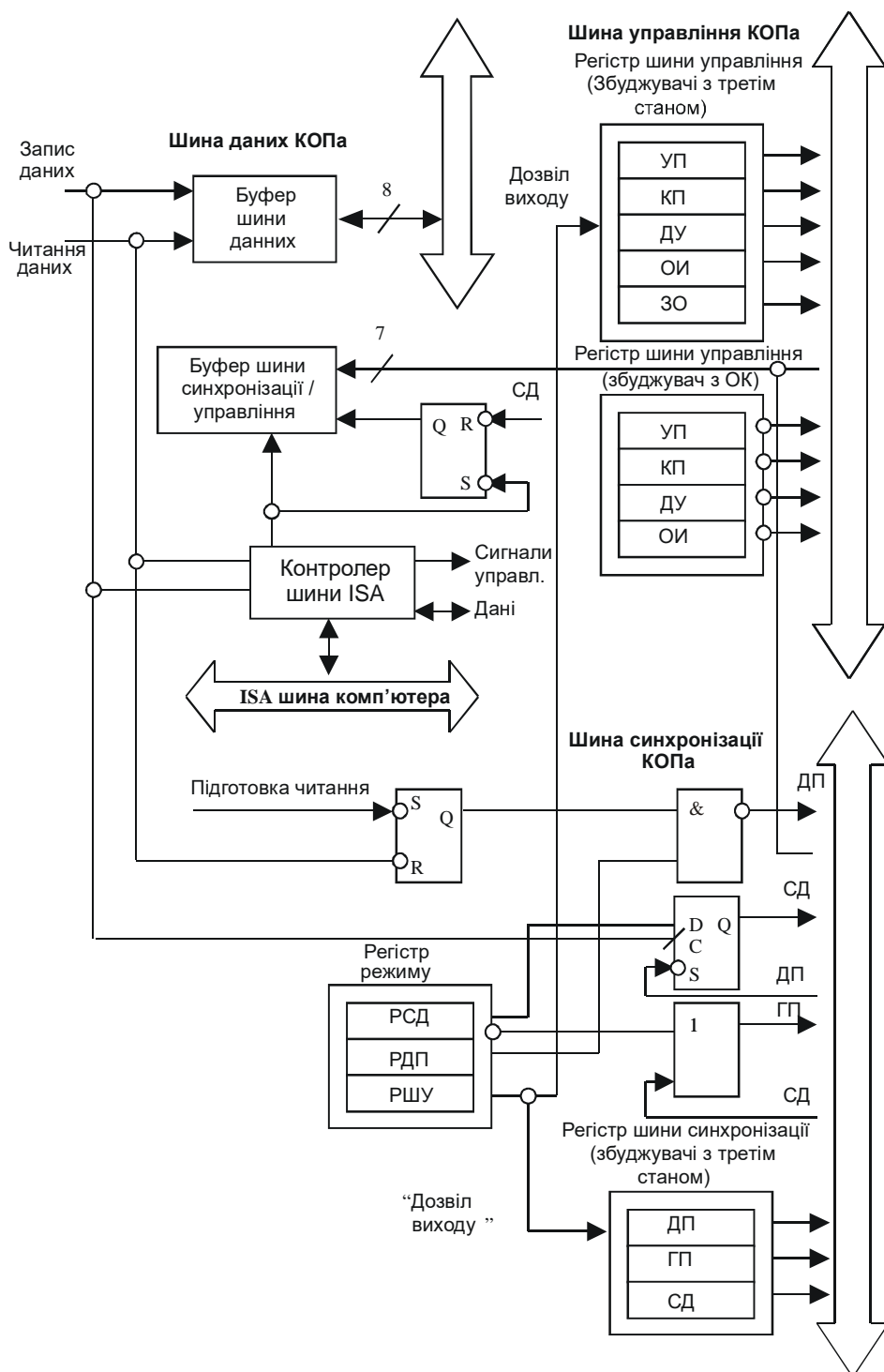
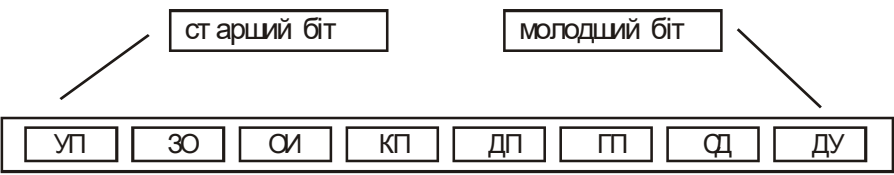
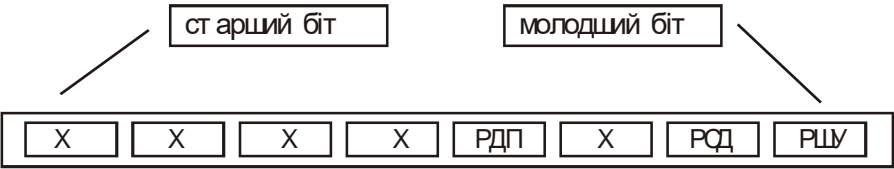
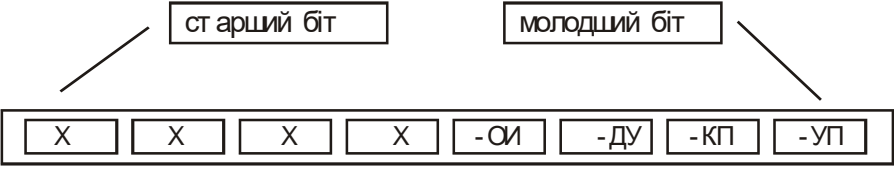


Рис. 7.4. Структурна схема інтерфейсу

Таблиця №7.7 (продовження)

Адреса та призначення порта	Призначення інформаційних бітів порта
ADR = \$ 2 , запис. Порт шини керування/ синхронізації КОПа.	 <p>ДУ...УП – біти, що відповідають значенням сигналів ДУ...УП на роз'ємі інтерфейсу</p>
ADR = \$ 3 , запис. Регістр режиму інтерфейсу.	 <p>РШУ – біт дозволу роботи вихідного регістра шини керування/синхронізації КОПа. Якщо даний біт встановлений в '1', то вихідний регістр шини керування/стану знаходиться в третьому стані. РСД – біт дозволу роботи ліній СД і ГП в автоматичному режимі (відповідно до протоколу КОПа). Якщо даний біт встановлений у '1', то дозволена робота лінії ГП, якщо в '0', то дозволена робота лінії СД і буфера шини даних на передачу в зовнішній пристрій. РДП – біт дозволу роботи ліній ДП в автоматичному режимі (відповідно до протоколу КОПа). Якщо даний біт встановлений у '1', то робота лінії ДП дозволена. X – незначущі біти.</p>
ADR = \$ 4 , запис. Регістр збуджувачів з СК шини керування КОП.	 <p>-ОИ, -ДУ, -КП, -УП – біти програмування збуджувачів з СК шини керування. Якщо якийсь із бітів встановлений в "1", то збуджувач відповідної лінії формує логічний "0". Якщо збуджувачі з СК шини керування не використовуються, то всі біти повинні бути встановлені в "0". X – незначущі біти.</p>
ADR = \$ 5 , запис. Команда підготовки лінії ДП до прийому.	Запис будь-якого числа за цією адресою приводить до установки на лінії ДП низького рівня сигналу (якщо робота лінії ДП дозволена установкою відповідного біта за ADR = \$ 3).

4. Програма для звертання до ЦАП/АЦП плати ET1270

У пункті 6 приведено текст тестової програми для роботи з платою **ЦАП/АЦП ET1270**. Програма може працювати в середовищі ОС **Windows 9x/ME**. Зовнішній вигляд головного вікна програми показано на рис. 7.5. Програма призначена для програмування пристрою **E7-14** на вимірювання певного параметру (параметрів) та отримування результатів вимірювання.

Команда, яка буде передана через інтерфейс **КОП** у пристрій при натисканні кнопки 2 ("Виконати", рис. 7.5), вводиться у текстове поле 1 ("Командний рядок"). Результат виконання команди (відповідь пристрою **E7-14**) відображається у текстовому полі 3 ("Результат виконання команди"). Код помилки відображується у статусному рядку 5. Код помилки описує стан плати **ET1270** або помилку, яка може виникнути в процесі роботи. Закінчення роботи програми здійснюється за допомогою кнопки 4.

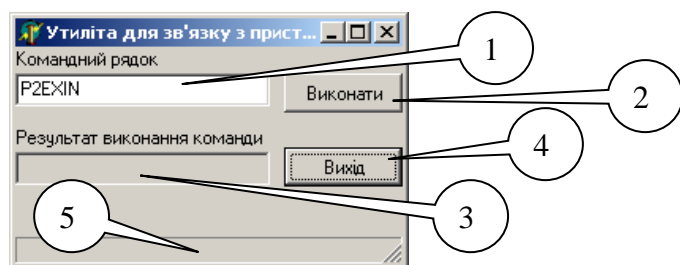


Рис.7.5. Головне вікно програми для зв'язку з вимірювачем імпедансу **E7-14** через плату **ET1270**.

Програмний код для реалізації зв'язку між платою та вимірювачем імпедансу міститься в процедурі **TfMain.btnCommandClick(...)**. Ця процедура здійснює програмування пристрою шляхом пересилання йому командного рядка, записаного у відповідне текстове поле головного вікна програми. Після цього від пристрою очікується відповідь у виді інформації про вимірний параметр. Алгоритм для здійснення вказаних дій включає наступні кроки:

1. Ініціалізація плати інтерфейсу **ET1270**.
2. Відкривання послідовного запиту.
3. Програмування пристрою **E7-14** на прийом команди.
4. Закривання послідовного запиту.
5. Пересилання команди до пристрою **E7-14**.
6. Відкривання послідовного запиту.
7. Програмування пристрою **E7-14** на пересилання даних до **ЕОМ**.
8. Читання даних із пристрою.
9. Закривання плати **ET1270**.

5. Завдання на лабораторну роботу

Написати програму (можна з використанням компонентів **VCL Delphi**), яка виконуватиме дії згідно варіанту (таблиця №7.8).

Таблиця №7.8

Варіант	Завдання
1, 6, 11, 16, 21, 26	Запрограмувати вимірювач імпедансу E7-14 на вимірювання опору резистора.
2, 7, 12, 17, 22, 27	Запрограмувати вимірювач імпедансу E7-14 на вимірювання ємності конденсатора.
3, 8, 13, 18, 23, 28	Запрограмувати вимірювач імпедансу E7-14 на робочу частоту 100 Гц для вимірювання опору резистора.
4, 9, 14, 19, 24, 29	Запрограмувати вимірювач імпедансу E7-14 на робочу частоту 1 кГц для вимірювання опору резистора.
5, 10, 15, 20, 25, 30	Запрограмувати вимірювач імпедансу E7-14 на робочу частоту 10 кГц для вимірювання опору резистора.

6. Програмні тексти

Текст файлу проекту.

```
program E7_14;

uses
  Forms,
  Main in 'Main.pas' {fMain},
  et8848 in 'ET8848.PAS';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfMain, fMain);
  Application.Run;
end.
```

Текст модуля головного вікна програми.

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls, ET8848;

type
  TfMain = class(TForm)
    eCmdLine: TLabelEdit;
    eResult: TLabelEdit;
    btnCommand: TBitBtn;
    btnExit: TBitBtn;
    sbBar: TStatusBar;
    procedure btnCommandClick(Sender: TObject);
    procedure btnExitClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  fMain: TfMain;

var I    : word;
    bl   : boolean;
    b    : byte;
    adr  : byte;

implementation

{$R *.dfm}

procedure TfMain.btnCommandClick(Sender: TObject);
var S : string;
    DL : word;
begin
  ET8848Init($120);
  sbBar.Panels[0].Text:=errorcode;

  // Ініціалізація ...
  adr:=0;
  ControlByte:=0;
  OutPort(base8848adr+3,ControlByte);
  ControlODByte:=0;
  OutPort(base8848adr+4,ControlODByte);
  ControlBusByte:=$FF;
  OutPort(base8848adr+2,ControlBusByte);

  deviceadr:=adr;
  DeActivateControlBus;
  SetODInterfClearSign;
  ReSetODInterfClearSign;
  NotEnableDataReciveSign;
  SetODDistControlSign;
```

```

ReSetODDistControlSign;
SetODDistControlSign;

try
    // відкривання послідовного запиту
    WriteComand($18);
    if ErrorF then raise Exception.Create('Послідовне опитування не відкрито.'#13+ErrorCode);
    // встановлення адреси
    WriteComand($20+deviceadr);
    if ErrorF then raise Exception.Create('Встановлення адреси.'+ErrorCode);
    // закривання послідовного запиту
    WriteComand($19);
    if ErrorF then raise Exception.Create('Послідовне опитування не закрито.'+ErrorCode);
except on E: Exception do
begin
    MessageDlg('Сталась помилка з описом:'#13+E.Message, mtError, [mbOK], 0);
    sbBar.Panels[0].Text:=errorcode;
    Exit;
end;
end;

OutString(eCmdLine.Text);
Sleep(1000);

ControlByte:=0;
OutPort(base8848adr+3,ControlByte);
ControlODByte:=0;
OutPort(base8848adr+4,ControlODByte);
ControlBusByte:=$FF;
OutPort(base8848adr+2,ControlBusByte);
deviceadr:=adr;
DeActivateControlBus;
SetODInterfClearSign;
ReSetODInterfClearSign;
NotEnableDataReciveSign;
SetODDistControlSign;
ReSetODDistControlSign;
SetODDistControlSign;
try
    // відкривання послідовного запиту
    WriteComand($18);
    if ErrorF then raise Exception.Create(ErrorCode);
    // встановлення адреси
    WriteComand($40+deviceadr);
    if ErrorF then raise Exception.Create(ErrorCode);
    ErrorCode:='Пристрій готовий.';
except
    ErrorCode:='Помилка ініціалізації пристрою : '+ErrorCode;
    MessageDlg('Сталась помилка з описом:'#13+ErrorCode, mtError, [mbOK], 0);
    sbBar.Panels[0].Text:=errorcode;
    Exit;
end;

SetReadFromDeviceMode;

// Читання даних із пристрою
sbBar.Panels[0].Text:='Читання...';
S:='';
for DL:=1 to 14 do S:=S+Chr(ReadData);
eResult.Text:=S;
sbBar.Panels[0].Text:=errorcode;

ET8848Close;
end;

procedure TfMain.btnExitClick(Sender: TObject);
begin
    Application.Terminate;
end;

end.

```

Рекомендована література

1. Руссинович М. Внутреннее устройство Microsoft Windows : Windows Server 2003, Windows XP и Windows 2000. Мастер-класс / М.Руссинович, Д.Соломон ; пер. с англ. – 4-е изд. – М : Издательско-торговый дом "Русская редакция" ; СПб : Питер, 2005. – 992 с.
2. Харт Джонсон М. Системное программирование в среде Windows / Джонсон М. Харт ; пер. с англ. – М. : Издательский дом "Вильямс", 2005. – 592с.
3. Рихтер Дж. Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows/ Дж.Рихтер ; пер. с англ. – 4-е изд. – СПб. : Питер; М.: Издательско-торговый дом "Русская редакция", 2001.– 752 с.
4. Рихтер Дж. Программирование серверных приложений для Microsoft Windows 2000 / Дж.Рихтер, Дж.Д.Кларк. Мастер-класс ; пер. с англ. – СПб. : Питер; М. : Издательско-торговый дом "Русская редакция", 2001. – 592с.
5. Саймон Р. Microsoft Windows 2000 API. Энциклопедия программиста / Р.Саймон. – М. : ДиаСофт, 2002. – 1085с.
6. Коноваленко І.В., Федорів П.С. Системне програмування у Windows з прикладами на Delphi. Навчальний посібник для технічних спеціальностей вищих навчальних закладів.– Тернопіль: ТНТУ.–2012.–320с.

Зміст

Вступ	3
1. Системне програмування	3
2. Лабораторний курс з основ системного програмування	3
Лабораторна робота №1	5
1. Принципи функціонування прикладних програм в середовищі ОС Windows	5
1.1. Система повідомлень в ОС Windows	5
1.2. Інтерфейс прикладного програмування	6
1.3. Структура прикладної програми з точки зору ОС	6
2. Приклад простої програми, написаної з використанням засобів Windows API	7
3. Завдання на лабораторну роботу	12
4. Текст програми	13
Лабораторна робота №2	15
1. Створення діалогового вікна прикладної програми засобами Windows API	15
2. Завдання на лабораторну роботу	20
3. Текст програми	22
Лабораторна робота №3	25
1. Побудова програми з інтерфейсом у виді піктограми на лінійці задач	25
1.1. Розробка основного виконавчого модуля програми	25
1.2. Розробка динамічної бібліотеки з діалоговим вікном	32
1.3. Звертання до функцій DLL з основного модуля	33
2. Завдання на лабораторну роботу	34
3. Програмні тексти	36
Лабораторна робота №4	44
1. Принципи написання зберігача екрану для ОС Windows	44
2. Завдання на лабораторну роботу	52
Лабораторна робота №5	54
1. Отримання інформації про ОС та встановлене обладнання	54
1.1. Отримання загальної інформації про ОС	55
1.2. Отримання інформації про пам'ять	57
1.3. Отримання інформації про дискові накопичувачі	57
1.4. Отримання даних про центральний процесор	57
1.5. Отримання списку процесів	58
1.6. Отримання списку вікон у системі	59
2. Завдання на лабораторну роботу	59
3. Програмні тексти	60
Лабораторна робота №6	65
1. Будова та характеристики плати інтерфейсу ET1270	65
1.1. Призначення та технічні характеристики	65
1.2. Роз'єми інтерфейсу	65
1.3. Робота з ЦАП і АЦП	66
1.4. Робота з цифровими портами	67
2. Програмне звертання до портів вводу/виводу у різних видах операційної системи Windows	68
2.1. Організація доступу до портів	68
2.2. Драйвер для доступу до портів з ОС Windows NT/2000/XP	69
2.3. Робота з драйвером програми на Delphi	70
3. Програма для звертання до ЦАП/АЦП плати ET1270	71
4. Завдання на лабораторну роботу	73
5. Програмні тексти	74

Лабораторна робота №7	89
1. Структура і призначення шини КОП	89
1.1. Шина даних (ШД)	89
1.2. Шина синхронізації (ШС)	89
1.3. Шина управління (ШУ)	89
1.4. Інтерфейсні повідомлення	91
1.5. Часова послідовність процесу синхронізації	92
1.6. Часова послідовність ідентифікації при запиті на обслуговування	93
2. Опис вимірювача імітансу E7-14	94
2.1. Призначення та основні технічні характеристики	94
2.2. Особливості роботи пристрою з КОП	96
3. Будова та характеристики плати інтерфейсу ET8848	97
3.1. Основні технічні дані і характеристики	97
3.2. Опис плати інтерфейсу	97
3.3. Адресний простір інтерфейсу	98
4. Програма для звертання до ЦАП/АЦП плати ET1270	100
5. Завдання на лабораторну роботу	101
6. Програмні тексти	102
Рекомендована література	104
Зміст	105